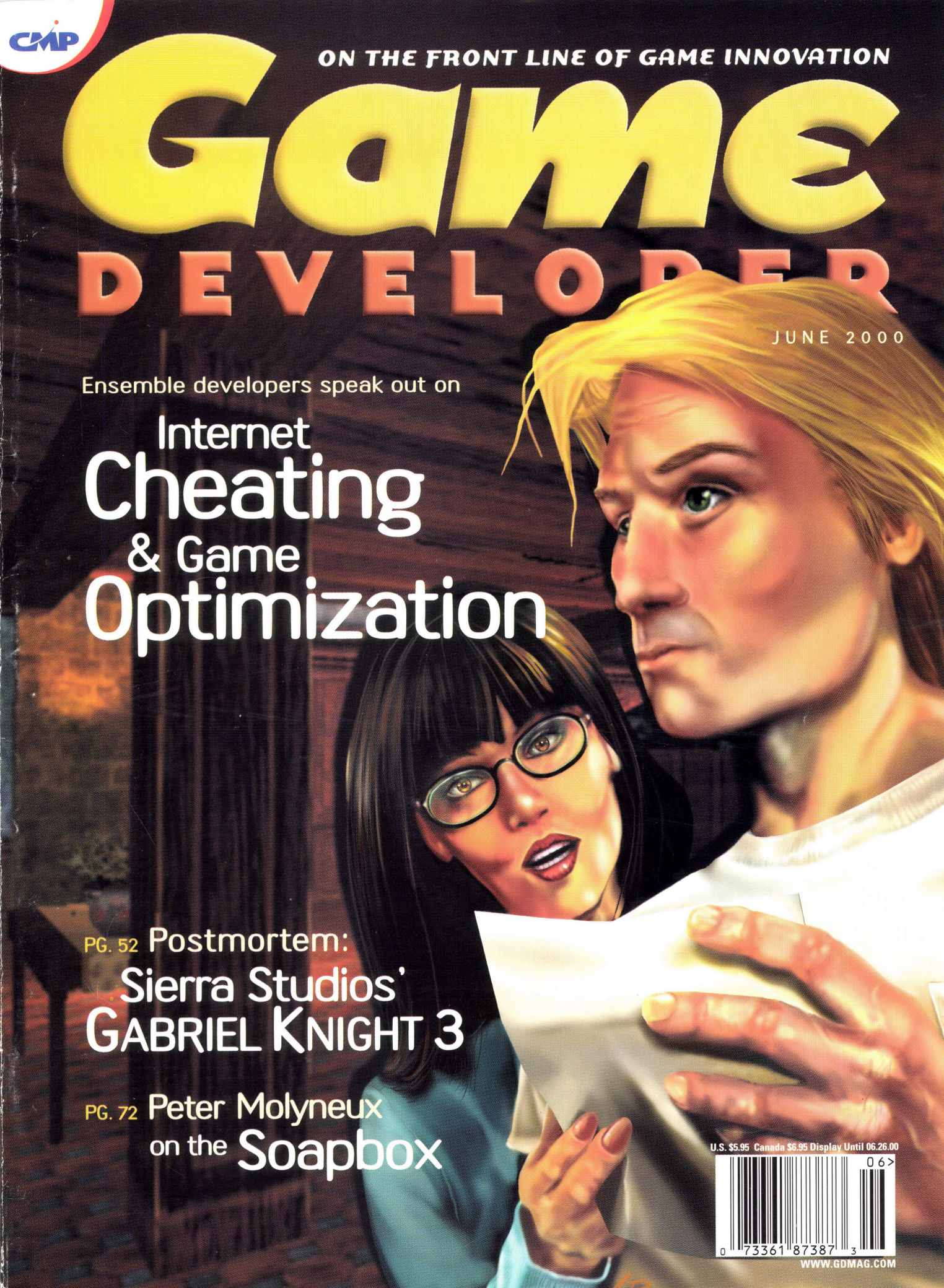
CMP

ON THE FRONT LINE OF GAME INNOVATION

GAME DEVELOPER

JUNE 2000



Ensemble developers speak out on

Internet Cheating & Game Optimization

PG. 52 Postmortem: Sierra Studios' GABRIEL KNIGHT 3

PG. 72 Peter Molyneux on the Soapbox

U.S. $5.95 Canada $6.95 Display Until 06.26.00

06>

0 73361 87387 3

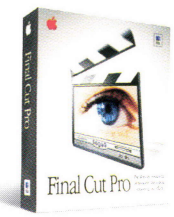
WWW.GDMAG.COM

Take a tour of a major

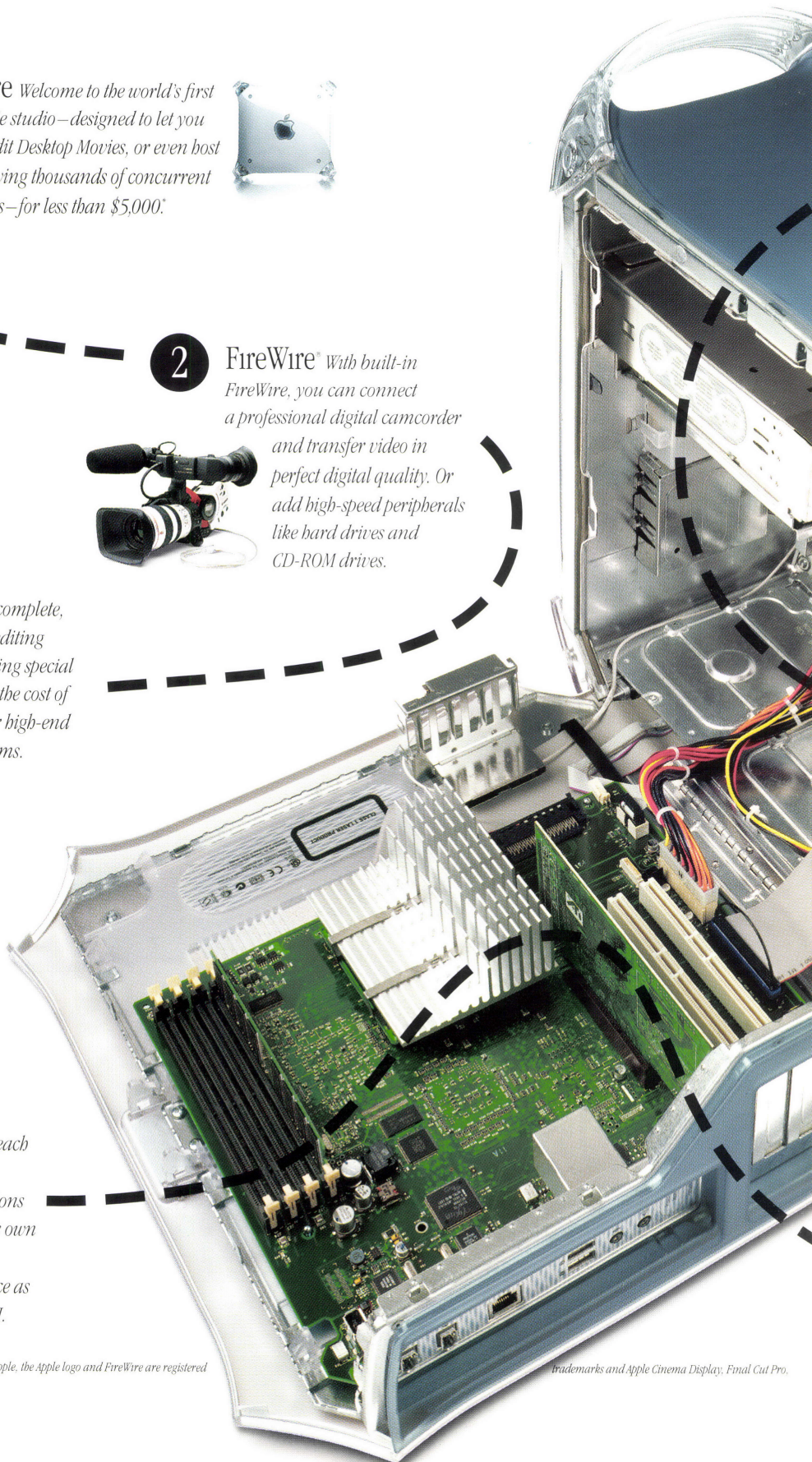
1 **Start Here** *Welcome to the world's first desktop movie studio—designed to let you create and edit Desktop Movies, or even host a website serving thousands of concurrent video streams—for less than $5,000.**

2 **FireWire®** *With built-in FireWire, you can connect a professional digital camcorder and transfer video in perfect digital quality. Or add high-speed peripherals like hard drives and CD-ROM drives.*

3 **Final Cut Pro™** *A complete, high-end digital video editing studio in a box, including special effects—at a fraction of the cost of other high-end systems.*

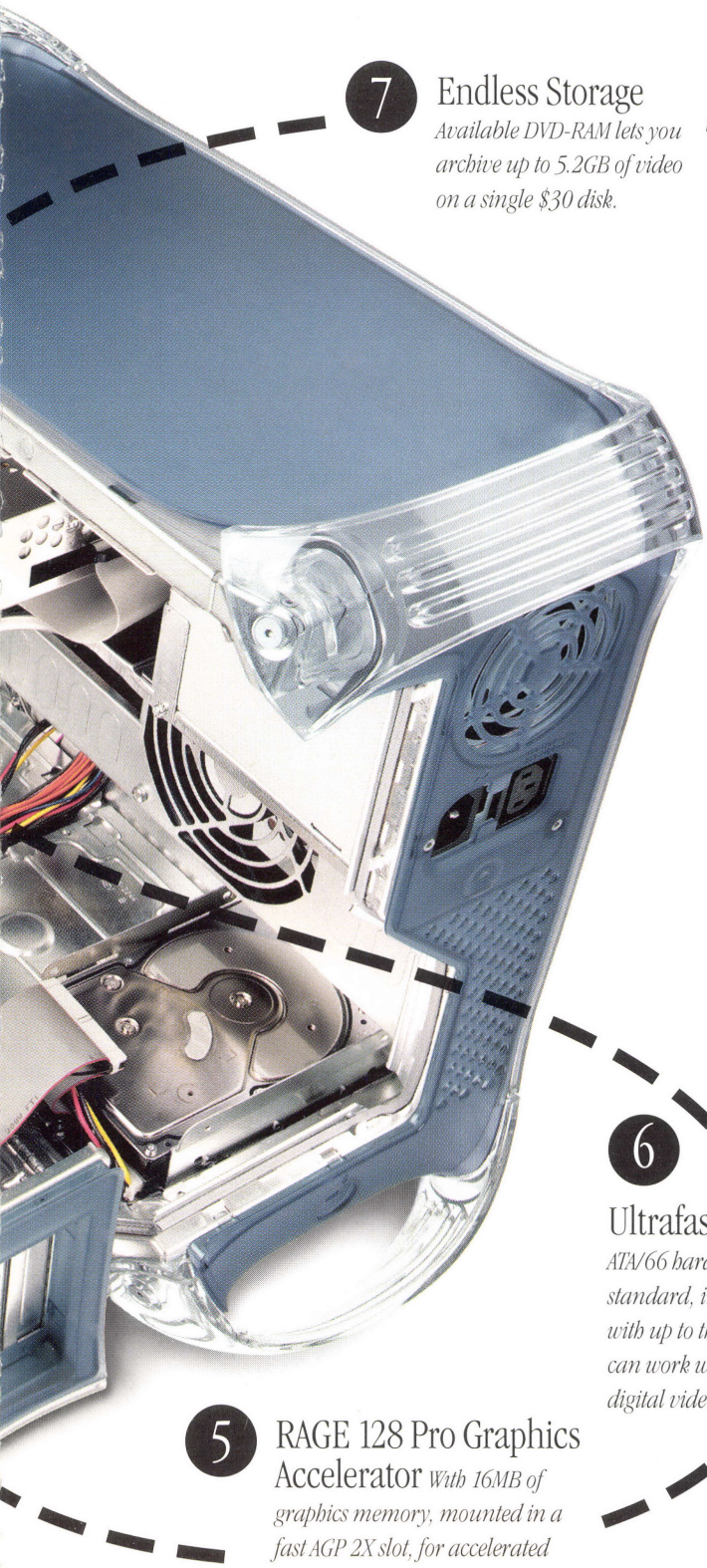


4 **G4 Chip with Velocity Engine™**
The first desktop processor to reach supercomputer speeds of one gigaflop (one billion calculations per second). Using six of Intel's own tests, the 500MHz G4 (with its 128-bit Velocity Engine) is twice as fast as the 800MHz Pentium III.*



©2000 Apple Computer, Inc. All rights reserved. Apple, the Apple logo and FireWire are registered trademarks and Apple Cinema Display, Final Cut Pro,

new movie studio.



7 Endless Storage
Available DVD-RAM lets you archive up to 5.2GB of video on a single $30 disk.

8 Screaming Streaming
Running QuickTime™ Streaming Server, your Power Mac™ G4 can host a website running thousands of concurrent video streams.



9 Apple Cinema Display™ *There's never been a movie screen like this in history. The Apple Cinema Display gives you the brightest, clearest, most inspiring place to work—a stunning 22 inches measured diagonally.*

6 Ultrafast Storage
ATA/66 hard disk up to 27GB standard, internally expandable with up to three drives—so you can work with up to six hours of digital video.

5 RAGE 128 Pro Graphics Accelerator *With 16MB of graphics memory, mounted in a fast AGP 2X slot, for accelerated rendering of 2D/3D images.*

Think different.™

*Power Mac, QuickTime, Think different and Velocity Engine are trademarks of Apple Computer, Inc. *Base configuration without camera. †Based on digital signal processing test scores.*

www.apple.com



Havok.com is the game technology wing of Telekinesys Research Limited.

havok.com

USA
Phone +1 408 376 4585
Fax +1 408 376 4586

Europe
Phone +353 1 677 8705
Fax +353 1 676 7094

www.havok.com

WARNING

PHYSICS. DEPLOY WITH CAUTION

SOFT BODIES - MAGNETISM - RIGID BODIES - FRICTION - FLUID - SMOKE
DON'T LET IT GET TO YOUR HEAD

CONTENTS

VOLUME 7, NUMBER 6

JUNE 2000

FEATURES

28 **How to Hurt the Hackers:** *The Inside Scoop on Internet Cheating and How You Can Combat It*

Ever wonder how to confront the never-ending struggle with cheaters who are determined to hack your game? Ensemble Studios' Matt Pritchard defines the rules of the cheating game and shares his insight on the methods and motivations of hackers, based largely on his experience with the AGE OF EMPIRES franchise.

Matt Pritchard

42 **Profiling, Data Analysis, Scalability, and Magic Numbers:** *Meeting the Minimum System Requirements for AGE OF EMPIRES II: THE AGE OF KINGS*

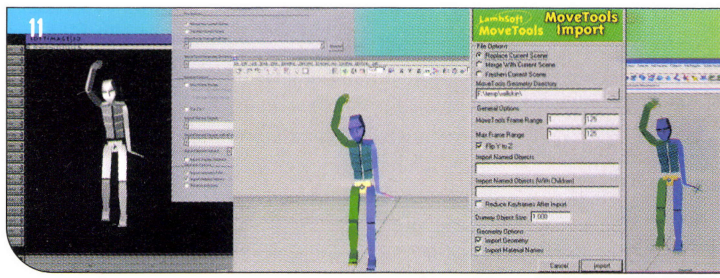
In the first of a two-part series, Ensemble Studios' Herb Marselas reveals the tips, tricks, tools, and pitfalls that went into raising AGE OF KINGS' performance profile so that it would run well on the minimum machine specification. If you're trying to get tomorrow's game to run on yesterday's hardware, this article can help.

Herb Marselas

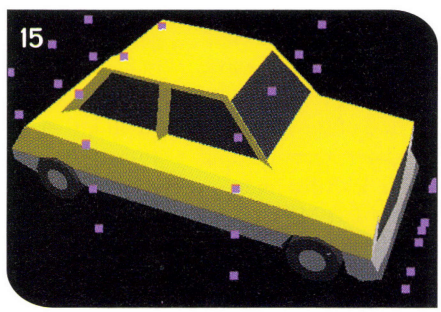
52 **Postmortem: Sierra Studios' GABRIEL KNIGHT 3**

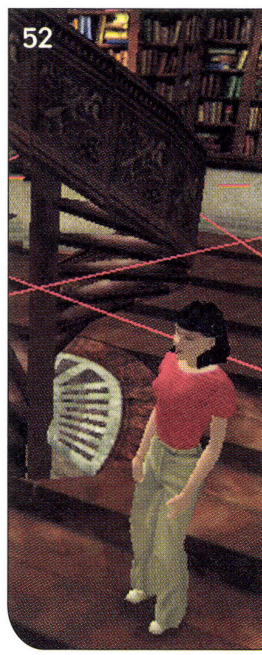
Redesigning a game engine midway through a project is never a task a team wants to undertake. But when the situation dictates it, you bite the bullet, just like the brave souls at Sierra Studios did. Learn how their development team overcame three years' worth of engineering tribulations and personnel turnover, and pumped out a massive three-disc adventure title.

Scott Bilas









COLUMNS

15 **GRAPHIC CONTENT** *Jeff Lander*

In This Corner...The Crusher!

23 **ARTIST'S VIEW** *Mel Guymon*

"The Best Laid Plans..."

72 **SOAPBOX** *Peter Molyneux*

Next-Generation Gaming: Raising the Stakes in the Publisher-Developer Relationship

COVER

Tito Pagan is an independent computer artist and 3D animator. He used Photoshop to create the cover art. You can view more of his work at www.titopagan.com.

DEPARTMENTS

4 **GAME PLAN** *Alex Dunne*

7 **FRONT LINE TOOLS** *Daniel Huebner*

Side Effects' Proceed, Lucent lip-synch, and more.

8 **INDUSTRY WATCH** *Daniel Huebner*

IGF results and the buzz on Sony broadband.

11 **PRODUCT REVIEW** *Mel Guymon*

Get the skinny on Lambsoft's Movetools.

63 **CREATIVE CAREERS**

71 **ADVERTISER INDEX**

Game Developer (ISSN 1073-922X) is published monthly by CMP Media Inc., 600 Harrison St., San Francisco, CA 94107, (415) 905-2200. Please direct advertising and editorial inquiries to this address. Canadian GST: 124513185. **SUBSCRIPTION RATE:** Subscription rate for the U.S. is $49.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: $69.95; all other countries: $99.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **POSTMASTER:** Send address changes to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. **CUSTOMER SERVICE:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call +1 (847) 647-5928 or fax +1 (847) 647-5972. Send payments to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. For back issues write to *Game Developer*, 1601 W. 23rd St. Ste. 200, Lawrence KS 66046-2703. Call toll-free in the U.S./Canada (800) 444-4881 or fax (785) 841-2624. All other countries call +1 (785) 841-1631 or fax +1 (785) 841-2624. Please remember to indicate *Game Developer* on any correspondence.

GAME PLAN

LETTER FROM THE EDITOR

Seumas McNally
1979–2000

For months I had planned to use this editorial to talk about the redesign of *Game Developer*. When you work on a magazine day in and day out for years, modifying superficial aspects of it like fonts and page layouts is a big deal to the publication's staff. Or at least these types of changes *seemed* like a big deal to us until March 21, the day we found out that Seumas McNally, president and lead programmer of Longbow Digital Arts, succumbed to Hodgkin's disease. How quickly perspectives change when confronted with the loss of someone you know.

The amount of time I knew Seumas was altogether too brief. Last fall an entry form for the Independent Games Festival from Longbow Digital Arts flashed across my desk with his name on it, and as we began evaluating the IGF submissions, it quickly became apparent that the company's entry, TREAD MARKS, would be one to beat. It sailed into the final round of the IGF and a letter went out to Longbow congratulating them on being chosen as a finalist and inviting them to the event last March at the San Jose Convention Center.

About a week before the IGF, the McNallys asked us whether a wheelchair would be available to them at the convention center. At the time I didn't know what it was needed for, but we made necessary arrangements and frankly we didn't think too much about it since we were in our pre-show crunch. But when I arrived at the Game Developers Conference and was told about Seumas's condition, it was almost too much for me to handle. None of the information on Longbow's entry form prepared me for my introduction to 21-year-old Seumas in his wheelchair, obviously suffering from the effects of a recent chemotherapy session. Everything about the situation stunned me: how young he was, how weak and tired he looked, and how difficult it must have been for Seumas and his family (who made up the rest of the Longbow team) to make the long trip from Providence Bay, Ontario, to California. But despite his obvious weariness, Seumas had a smile for everyone and seemed genuinely happy to be at the event.

TREAD MARKS took three awards at the IGF: Best Design, Best Programming, and the Grand Prize. When the McNallys came up to the stage to accept their first award for game design, Seumas nearly collapsed from weakness as he got out of his wheelchair and tried to step up onto the 18-inch riser to accept the award. His dad, Jim, caught him in time and the whole family came over to the podium to accept the award. In his acceptance speech, Seumas talked in a quiet voice about working on the game while battling cancer and revealed that the goal of completing the game had helped him continue on. A number of people in the audience began crying. The McNallys came up to receive their next two awards shortly afterwards and when the event was over, seeing the McNallys — and most of all Seumas — smiling and laughing gave me an indescribable feeling. It's a moment I'll never forget.

The next day, Saturday, Seumas collected all his strength and presented a lecture with Jonathan Blow on terrain rendering systems. Seumas talked from his wheelchair, often stopping to catch his breath. On Sunday and Monday, Seumas remained in bed in the McNallys' hotel room. Many of his friends came by to visit him and even though Seumas was too short of breath to talk much, he appreciated the company.

Seumas had a hard time getting home and his health continued to decline. Jim McNally told me that Seumas was determined to remain alive until the end without sedation. "Unfortunately he suffered considerably, but as was his nature, he never complained," Jim said.

Seumas died on March 21 at 2:00 P.M. EST. I, like many others in the game development community, will not forget him, nor the example he set.



Let us know what you think. Send e-mail to editors@gdmag.com, or write to Game Developer, 600 Harrison St., San Francisco, CA 94107

ON THE FRONT LINE OF GAME INNOVATION

Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.2228 w: www.gdmag.com

Publisher
Jennifer Pahlka *jpahlka@cmp.com*

EDITORIAL
Editorial Director
Alex Dunne *adunne@sirius.com*
Managing Editor
Kimberley Van Hooser *kvanhoos@sirius.com*
Departments Editor
Jennifer Olsen *jolsen@sirius.com*
News & Products Editor
Daniel Huebner *dan@gamasutra.com*
Art Director
Laura Pool *lpool@cmp.com*
Editor-At-Large
Chris Hecker *checker@d6.com*
Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@infinexus.com*
Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook Verant Interactive
Susan Lee-Merrow Lucas Learning
Mark Miller Group Process Consulting
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt Microsoft

ADVERTISING SALES
National Sales Manager
Jennifer Orvik e: *jorvik@cmp.com* t: 415.905.2156
Account Executive, Western Region & Asia
Mike Colligan e: *mcolligan@cmp.com* t: 415.356.3486
Account Executive, Eastern Region & Europe
Afton Thatcher e: *athatcher@cmp.com* t: 415.905.2323
Sales Associate/Recruitment
Morgan Browning e: *mbrowning@cmp.com* t: 415.905.2788

ADVERTISING PRODUCTION
Senior Vice President/Production Andrew A. Mickus
Advertising Production Coordinator Kevin Chanel
Reprints Stella Valdez t: 916.983.6971

CMP GAME MEDIA GROUP MARKETING
Marketing Manager Susan McDonald
Product Marketing Manager Darrielle Sadle
Field Marketing Manager Jennifer McLean
Marketing Coordinator Scott Lyon

CIRCULATION
BPA INTERNATIONAL
Game Developer magazine is BPA approved
Vice President/Circulation Jerry M. Okabe
Assistant Circulation Director Kathy Henry
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Pam Santoro

INTERNATIONAL LICENSING INFORMATION
Robert J. Abramson and Associates Inc.
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

CORPORATE
President & CEO Gary Marshall
Corp. President, Business Tech & Channel John Russell
President, Business Technology Group Adam Marder
President, Specialized Technology Group Regina Ridley
President, Channel Group Pam Watkins
President, Electronics Group Steve Weitzner
General Counsel Sandra L. Grayson
Vice President, Creative Technologies Johanna Kleppe
General Manager, CMP Game Media Group Greg Kerwin

CMP

awesome performance truly multi-platform open and flexible architecture

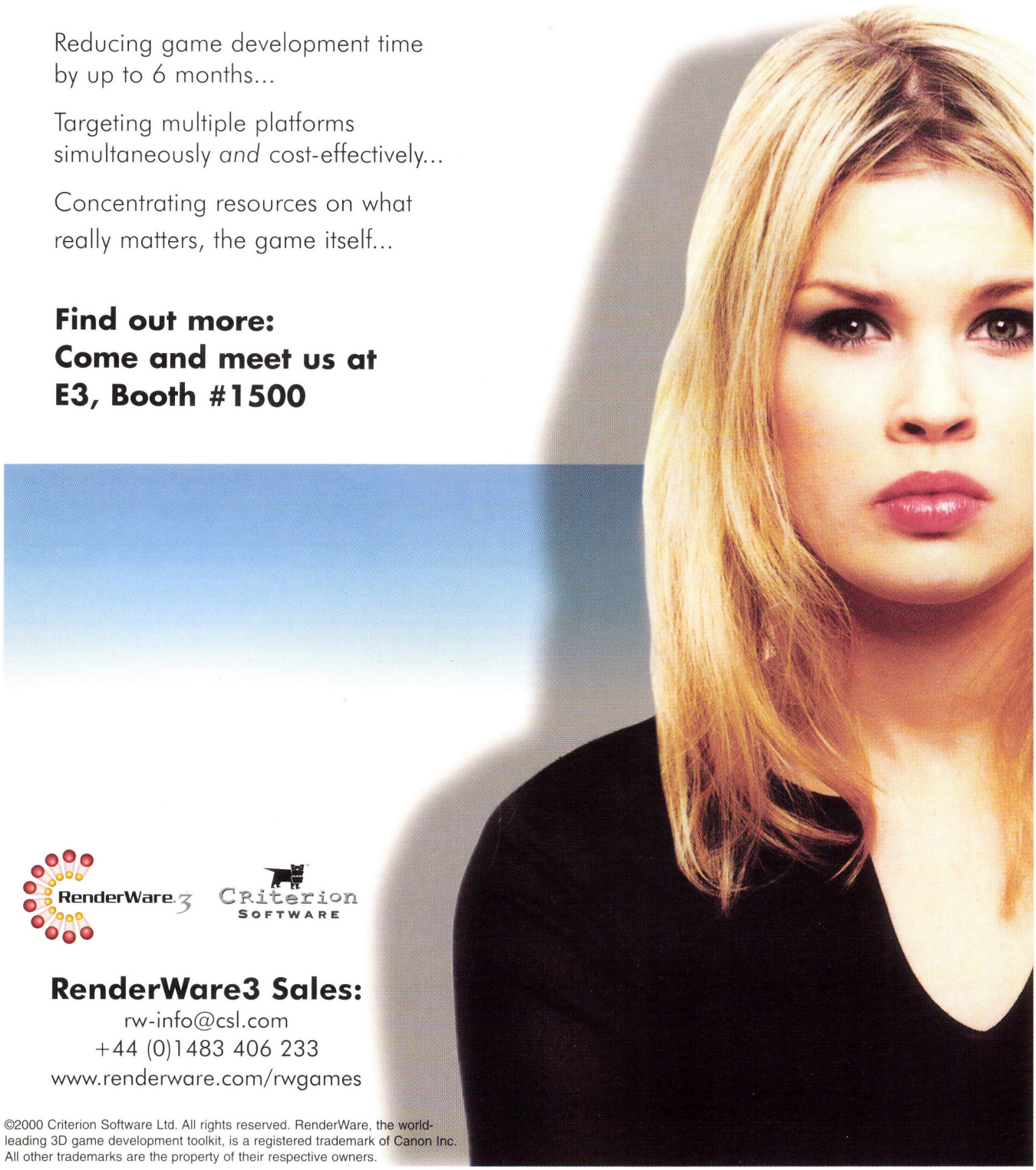
This *is* the real world! RenderWare3...

Reducing game development time by up to 6 months...

Targeting multiple platforms simultaneously *and* cost-effectively...

Concentrating resources on what really matters, the game itself...

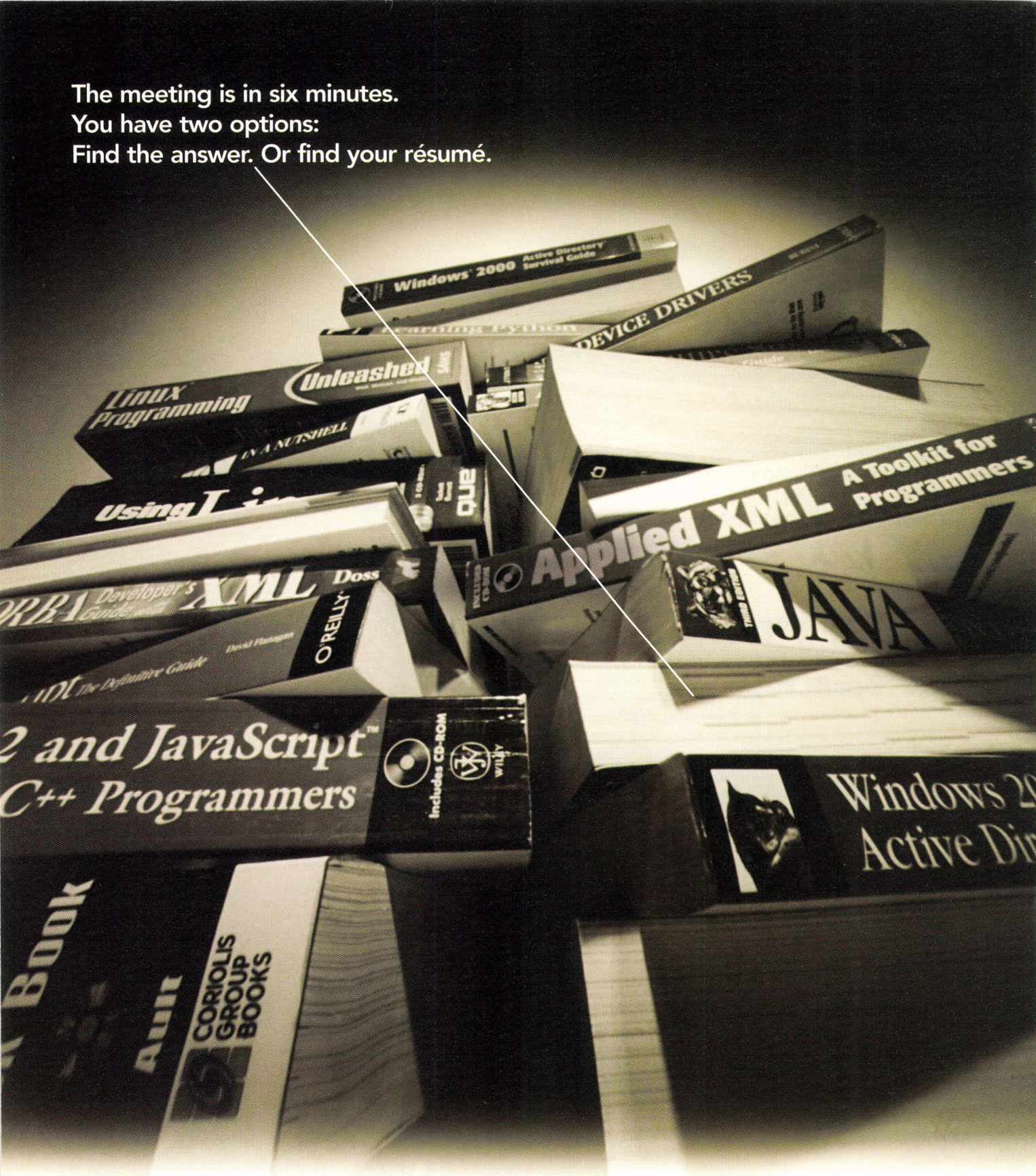
Find out more:
Come and meet us at
E3, Booth #1500



RenderWare 3 Criterion Software

RenderWare3 Sales:
rw-info@csl.com
+44 (0)1483 406 233
www.renderware.com/rwgames

©2000 Criterion Software Ltd. All rights reserved. RenderWare, the world-leading 3D game development toolkit, is a registered trademark of Canon Inc. All other trademarks are the property of their respective owners.



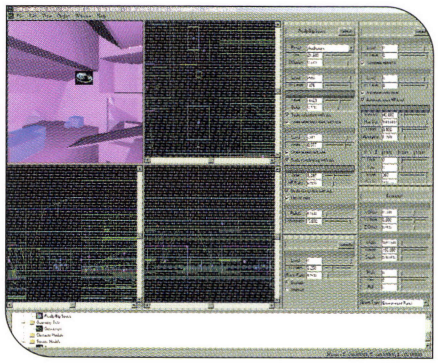
When you gotta find a needle in a bookstack. Mountains of manuals. Rivers of reference material. And the boss is hot for answers. Feeling a little pressure? Now you don't have to. Just go to ibooks.com℠ — the digital bookstore of the future. We have the best IT reference books available online. Instantly find the digital book *and* the answer you need. Easily build your personal digital bookshelf. It will always be there for you — when you gotta have it.

ibooks.com℠
information. unbound.

©2000 ibooks.com All rights reserved.

FRONT LINE TOOLS

WHAT'S NEW IN THE WORLD OF GAME DEVELOPMENT | *daniel huebner*



EAGLE | Creative Labs | developer.soundblaster.com

THE EAGLE HAS LANDED

Creative's Environmental Audio Graphical Librarian Editor, less ponderously known as EAGLE, is a simple sound modeling tool designed to help game developers take full advantage of Creative's EAX environmental audio technology. EAGLE lets sound designers create 3D models to import actual game level geometry and assign environment and obstacle properties to rooms and areas of the map. EAGLE then applies these effects interactively during game play. EAGLE can be obtained at no charge from Creative's web site.

SIDE EFFECTS' PROCEED

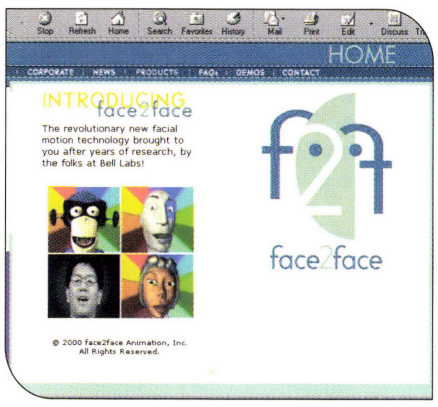
SIDE EFFECTS SOFTWARE

Billed as a content-to-code solution, Side Effects Software's Proceed (codename) combines the film-level authoring capabilities of the company's Houdini package with run-time engines for both motion and particles, plus automatic code generation. Proceed offers a collection of NURBS, Bézier, and polygonal surface and solid modeling operations, including surfacing, fitting, cutting, projection, instancing, deformation, dynamics, H-splines, and conversion. Choreography through motion layering and manipulation is handled by Proceed's Channel Operators technology, allowing nondestructive motion editing suitable for motion capture, motion-on-motion compositing, lip-synching, facial animation, motion management for large projects, real-time gesture capture, audio editing, and audio synthesis. Proceed can automatically generate code for both motion compositing and particle systems. The package provides a high-level interface, but operators can be created and customized using a development kit available at no extra charge. Proceed supports Windows NT 4.0 with SP3 or later, Windows 2000, IRIX 6.2 or later, and Red Hat Linux 6.0. Proceed will ship during the second quarter of this year with the option of either a fixed-price or royalty-based pricing scheme.

PROCEED | Side Effects Software | www.sidefx.com

MO-CAP AT FACE VALUE

Lucent Technologies is licensing a new automated lip-synchronization system aimed at the television, Internet, and game development markets. Face 2 Face uses facial motion analysis to capture head and face movement without the use of markers, stickers, or headgear. The system converts video to an RGB .MOV format which is then processed to identify facial definition points. The output is a frame-by-frame capture of the definition points, which can then be mapped to an animation. Based on the emerging MPEG-4 standard, Face 2 Face lets animators transparently adapt content to multiple media. The system operates with software plug-ins for multiple animation packages on Windows NT and IRIX, and requires S-VHS or better video for capture. Face 2 Face is available in downloadable formats with pricing based per frame of processed video.



FACE 2 FACE | Lucent Technologies | www.f2f-inc.com



Moss Lake by Terry Stoeger, Alias|Wavefront

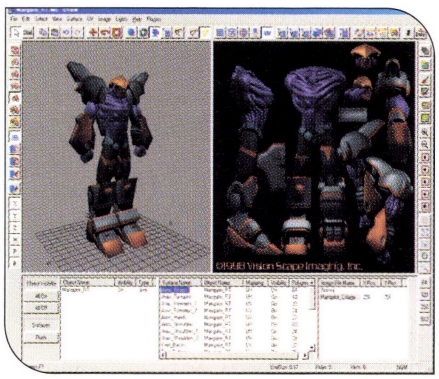
TRAVELING PAINT EFFECTS

Alias|Wavefront has a new plug-in for Adobe After Effects 4.1 and Maya Fusion 2 that allows motion graphic artists to take advantage of Maya Paint Effects technology within desktop compositing applications. Maya Paint Effects users can use brushes created for any Paint Effects product, including Maya itself, to create animated paint strokes of plants, fibers, and other natural media along a mask, Bézier path, or motion path. The Maya Paint Effects plug-in is available for $499 for Windows-based computers.

PAINT EFFECTS PLUG-IN | Alias|Wavefront | www.aliaswavefront.com/entertainment

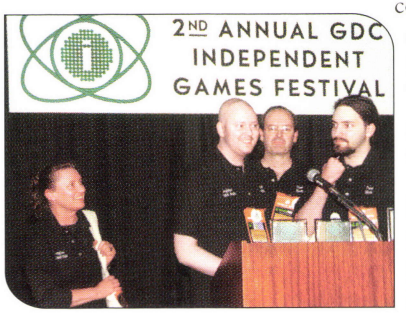
UVIEW 2.1 EXPANDS ITS FRONTIERS

Originally developed around Newtek's Lightwave 3D file format, Cinegraphics' Uview 2.1 has been expanded to work with every major 3D animation package. Uview is a stand-alone application designed to ease UV texture mapping. Enhancements include the addition of groupings for vertices and polygons, a texture bounding feature, UV mirroring to form symmetrical objects, and improved translation and rotation controls. Uview can output to proprietary game engines as well as commercial 3D packages. Prices are $599 for Intel-based Windows 98/NT systems and $729 for DEC Alpha Windows NT.



UVIEW 2.1 | Cinegraphics | www.cinegraphics.com

Indie Games Festival. The Second Annual Independent Games Festival was once again a highlight of the Game Developers Conference this past March. The conference crowds were most impressed with THE RIFT from Seattle's Thrushwave Technologies, giving the space strategy game the Audience Award. The prize for Best Audio went to BLIX, a retro-styled puzzler from New York's Station Blix, while RPG KING OF DRAGON PASS took home the award for Best Visual Art. The stars of the festival, though, were Seumas McNally and Longbow Digital Arts. The festival's judges honored Longbow's TREAD MARKS, a futuristic tank battle game developed by a unique collaboration between two brothers and their parents, with three of the festival's top awards. The game earned awards for Best Programming, Best Game Design, as well as the Grand Prize for best overall game. The audience at the IGF awards ceremony was clearly moved by Seumas McNally's acceptance speech, in which he talked about his fight against cancer. Seumas lost that battle just 11 days later. The Independent Games Festival has since decided to honor the memory of Seumas McNally by renaming the festival's highest award, now to be known as the Seumas McNally Grand Prize. "By renaming the top IGF award in his honor, we hope to uphold that same spirit that Seumas so freely displayed to those who shared his passion for game development," said IGF founder and chairman Alex Dunne.



The McNally family accepts one of Longbow Digital Arts' three awards at the IGF.

More Changes to the Graphics Industry Landscape. ATI's purchase of ArtX is not the last word in graphics chip industry changes. Struggling chipmaker 3dfx is investing $186 million, in the form of 15.6 million 3dfx shares and options, to acquire Gigapixel Corporation, recently spurned by Microsoft as the graphics provider for the X-Box. 3dfx hopes to exploit Gigapixel's core technologies, those that reduce power and memory consumption in 3D acceleration, in both traditional accelerators and products related to possible new 3D markets like cell phones and PDAs. As part of the deal, Gigapixel CEO George Haber will join the 3dfx board of directors.

While everyone else seems to be a in buying mood, S3 Inc. recently sold off its graphics chip business in order to focus on its burgeoning Internet device business. The company sold its graphics chip business to a newly formed joint venture with Taiwan's VIA Technologies for $323 million in cash and securities.

Learning Company up for Sale. Mattel is looking to unload The Learning Company less than a year after purchasing it. Mattel is separating The Learning Company from the remainder of Mattel Interactive and has hired Credit Suisse Boston to sell the software maker that it spent $3.6 billion to acquire last May. The purchase had been part of former Mattel CEO Jill Barad's strategy to expand Mattel into electronic toys and videogames, but accounting problems and huge losses related to the acquisition led instead to Barad's resignation. At the time of the acquisition, Mattel had expected The Learning Company to contribute $50 million annually to its bottom line, instead the company posted losses of $183 million in the fourth quarter and $206 million for the year. Mattel began treating the unit as a discontinued operation effective March 31. The *Wall Street Journal* estimated that the sale price for The Learning Company would range from $500 million to $1 billion, depending on included licenses. Havas, Knowledge Universe, and Infogrames were said to be interested buyers.

Sony Broadband Consolidation. Sony is making moves to strengthen its online content offerings, in part by grouping Sony Music Entertainment and Sony Pictures Entertainment into a single new holding company called Sony Broadband Entertainment. The new company is charged with finding ways to exploit the cross-market potential of Sony's entertainment holdings in conjunction with Sony's planned broadband network. The first effect of this reorganization was the departure of Lisa Simpson, president of Sony Online Entertainment. Simpson left the company to take a position with the CBS Internet Group. Former 989 Studios head Kelly Flock is said to be Sony's top choice to take on the division in the role of CEO.

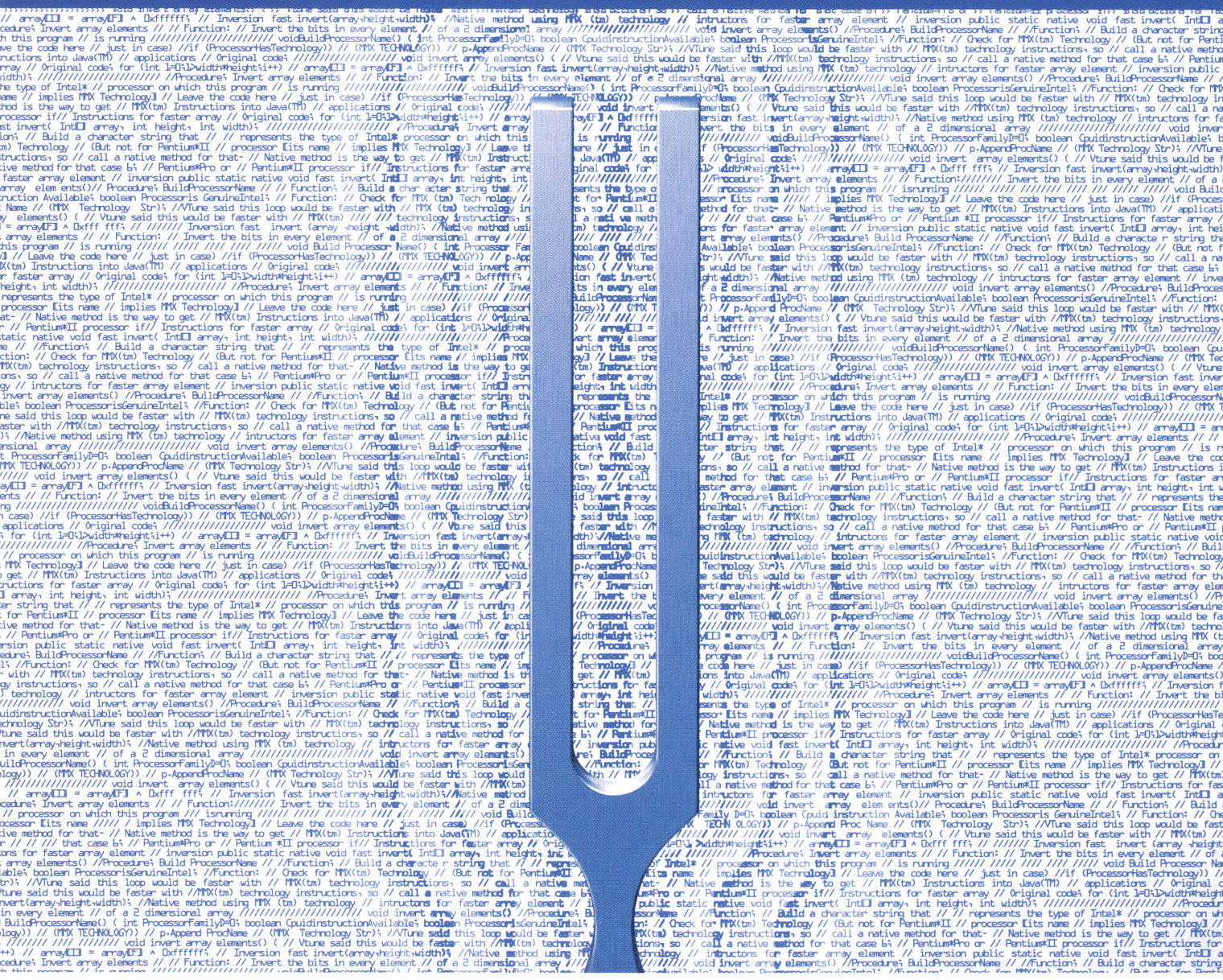
Aureal in Turmoil. Poor fourth-quarter financials were only the beginning for Aureal, as the company faced the mass resignation of all executive officers and senior staff members at the end of March. Aureal's fourth-quarter operations generated a net loss of $9.5 million, a sizable increase from last year's fourth-quarter loss of $3.5 million. Losses for the year reached $26.9 million, compared with losses of $18.5 million in 1998. Aureal attributed some of this to $6.4 million in legal fees related to its patent suit with Creative. Resignations followed the financials by a day, with president and CEO Kip Kokinakis, CTO Scott Foster, CFO David Domeier, and COO and general counsel Brendan O'Flaherty announcing their departures along with the entire senior staff. The Aureal board of directors has filed for Chapter 11 insolvency protection while it considers options ranging form appointing new turnaround management to a liquidation of assets and cessation of operations.

UPCOMING EVENTS CALENDAR

MACWORLD EXPO
JACOB K. JAVITS CONVENTION CENTER
New York, N.Y.
July 18–21, 2000
Cost: $25–$1,295
www.macworldexpo.com

SIGGRAPH 2000
ERNEST N. MORIAL CONVENTION CENTER
New Orleans, La.
July 23–28, 2000
Cost: $65–$960 (member and student discounts available)
www.siggraph.org/s2000

your code runs best
when it's in tune with the latest intel processors.



nothing tunes code like the latest vtune™ analyzer and intel® compilers. introducing the vtune performance analyzer v4.5 and intel c/c++ and fortran compilers. the vtune performance analyzer not only identifies bottlenecks and performance enhancement opportunities. it features a built-in code coach that uses the intelligence inside intel compilers to suggest optimization possibilities. and intel compilers help get the most out of intel's latest processors. what's more, they plug directly into microsoft visual studio®. so get your hands on these intel software performance tools. because who knows more about optimizing for intel processors than intel?



download a free, 30-day evaluation → *http://developer.intel.com/vtune*

intel

Intel is a registered trademark and VTune is a trademark of Intel Corporation. © 2000 Intel Corporation.
*All other brands and trade names are the property of their respective owners.



WHEN REALTIME IS OF THE ESSENCE

- **Empower your unique ideas with MultiGen Creator**
- **Take advantage of MultiGen Creator's sophisticated realtime 3D modeling toolset**
- **Deliver powerful and optimized 3D content to many platforms**
- **Produce models and environments with an artist's eye and a programmer's structure**
- **Use the application realtime 3D professionals prefer**
- **Author ready-to-run virtual worlds for the most demanding entertainment, urban simulation, and visual simulation applications**

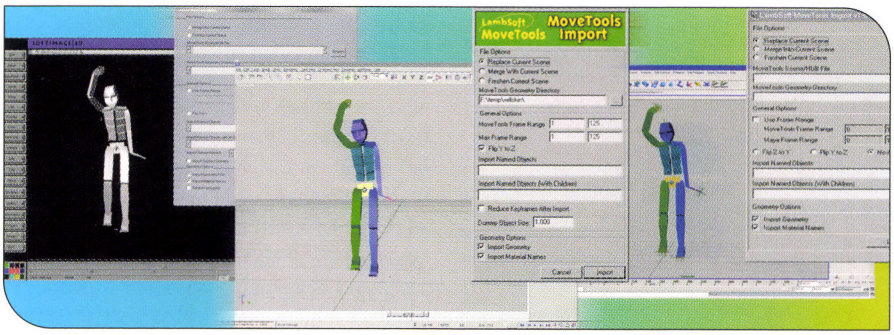
For more information contact MultiGen-Paradigm at:

Phone: 408 261 4100
Fax: 408 261 4103
email: sales@multigen.com
http://www.multigen-paradigm.com

Ad reference: CreatorGDmag00

Constructed in MultiGen Creator / Imagery Provided by Dark Suns Studios, LLC.

MultiGen • Paradigm
THE REALTIME 3D COMPANY



Lambsoft's Movetools

by mel guymon

Back in October 1999, Lambsoft quietly released a flagship product called Movetools, a file converter for 3D Studio Max, Softimage, and Maya. Since its release, artists and animators in the game industry have been flocking to the product, which is rapidly gaining a reputation as the final word in file translation. This month, I had a chance to test drive the software and put it through its paces, and as you read on, you'll see I was pretty pleased with the results.

Why Do You Need a Converter?

To artists and animators with experience in the industry, the answer to this question is obvious and comes in two parts. First, one classic problem in game development is that there is usually no single piece of software which encapsulates all the functionality needed by developers. Typically, studios end up using one platform for animation, another for modeling and texturing, and yet a third for exporting the data into the game engine. Mixing and matching tools can be a tempting solution, but the trade-off is substantial, since time will inevitably be wasted managing the different naming conventions, file formats, and animation protocols specific to each tool.

The second part of the answer has to do with personnel. While the feature sets for each of the big three platforms have been converging, artists and animators still tend to be experts in only one tool. The ideal scenario, then, is to let animators work in the environment in which they are most experienced, which means more pressure on everyone else to incorporate those tools with which they are familiar.

It's clear that having the ability to integrate content seamlessly from multiple software packages is a powerful benefit not to be undervalued. This is precisely the functionality that Movetools promises to deliver.

Methodology

Basically, Movetools works as a plug-in for each of the three platforms. The Movetools file structure, which is based on Alias|Wavefront's venerable .OBJ format, serves as the hub of a wheel through which each of the three platform plug-ins communicates. In order to get files from Maya into Softimage, for example, the Maya content is exported into an intermediate Movetools scene database, which is then imported directly into Softimage with little or no loss of information. The concept of how this works is further reinforced by the nomenclature, as the Movetools files are compressed into .HUB files and the individual platform licenses are termed "spokes."

One of the few problems I had with the implementation of this process is that each of the spokes must be purchased and licensed separately. This means that you end up buying at least two licenses for the software, one for the program you export from and one for the program you import to. Furthermore, the spokes are licensed to only a single machine, not to the software's hardware lock, which means that the spokes are not portable between machines.

THE .HUB FILE. After Movetools exports the data into a Movetools scene database, the .OBJ and scene files are further compressed into a single .HUB file, making the data footprint smaller and the file management task simpler. The data within the .HUB file is readable with Winzip, so that the interim step data format is readily accessible (Figure 1).

The fact that the .HUB file is an external format enables modelers, texture artists, and animators to work independently from each other, all within the platform most appropriate for their task. This can be extremely useful when compiling large animation databases, which historically require the animator to work with the finished or near-finished model. In Movetools, this is made possible through the use of a Freshen modifier in the import process.

FRESHEN. The Freshen modifier is an option that allows the artist to update his or her scene with a set of animation data without destroying the current scene. This means that animation data from any external program, once it has found its way to the Movetools scene database, can be applied to the current character. This has particular relevance for game development, since each new character need not contain a completely unique set of animations. Furthermore, this process allows multiple animators to work simultaneously on the same character, since a single version can be "freshened" from multiple sources. Finally, it makes the process of applying large volumes of motion capture data extremely efficient.

Movetools' Capabilities

From an animation and modeling standpoint, Movetools supports most of the basic functionality for game development. Polygonal and NURBS-based geometry, dummy nodes, grouping, and null objects

AUTHOR'S BIO | *Mel is rapidly discovering he's a gadget freak and is becoming addicted to doing reviews of cool plug-ins and such. Send all inquiries and free stuff to mel@infinexus.com.*

★★★★★ excellent
★★★★ very good
★★★ average
★★ fair
★ don't bother

are all supported, but even more impressive is the fact that the hierarchies and bone structures for each of the platforms are preserved. Position, rotation, and scaling transform values are sampled once per keyframe, regardless of the number of data points generated by hand or through mocap. This ensures that the curvature and interpolation values in the animation controllers are preserved, but results in a very high data frequency. Shape shifting and morph-target animation are not currently supported in Movetools, however, due to the dissimilar ways in which the individual platforms accomplish these tasks.

Geometry and texture coordinates are currently transferred. Textures (and materials) may be reapplied and will appear the same, since the UV coordinates are transferred as well. Various types of camera and lighting information come through as well. The camera lens type and field of view, as well as the light type and shadow parameters, are all supported.

Individual Interfaces

Because it's a plug-in and not a standalone, Movetools depends on the native interfaces of the programs it works with to accomplish its task. I found this to be refreshing, since I didn't have to learn an entirely new external interface. And though the basic functionality is the same between platforms, the implementation is unique to each.

MAYA. In Maya, Lambsoft capitalizes on the Maya Embedded Language (MEL) to create the Movetools GUI, which is particularly clean and straightforward. In addition to the standard options, the Maya spoke allows for the importing and exporting of skinning parameters for all objects, which is particularly useful in character animation. Additionally, the spoke can convert the coordinate system of the data by flipping the Z to Y coordinate or vice versa. This is useful when working in a package that has a Z-up coordinate system such as 3D Studio Max.

Although most of the functionality of Maya is kept intact, there are some exceptions. Objects built using patches must first be converted to NURBS or polygon objects prior to export. Also, only those materials in the current material library will be accessed, and then only if there is a material name match. Finally, while the polygon smoothing data can be exported without a problem, it's not accessible on import.

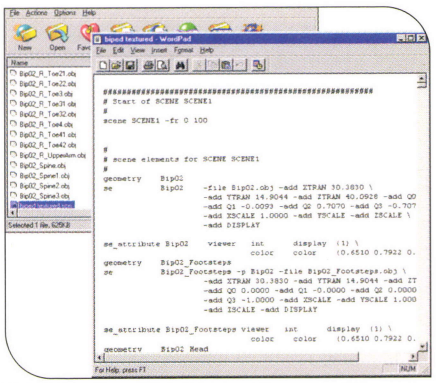


FIGURE 1. The .HUB file, readily accessible through Winzip.

SOFTIMAGE. In Softimage, the Movetools spoke is accessed through the Custom Tools interface. The basic functionality is the same as in Maya with geometry and animation import and export and manipulation of coordinate systems. There are, however, some key differences in the way Softimage handles data, and these are carried through to the Movetools implementation.

First, Softimage is unique in the way it handles naming conventions since objects can be named using a prefix. Objects built using patches must be first converted to NURBS or polygonal objects prior to export. Also, only those materials in the current material library will be accessed, and only if there is a material name match. Finally, while the polygon smoothing data can be exported without a problem, it is not accessible on import.

3D STUDIO MAX. In Max, the Movetools spoke is accessed via the file export and import drop-down menu. The basic functionality is identical to Maya and Softimage with a few notable differences. In Max, animations imported from Movetools scene databases can have their keyframes reduced automatically. This feature uses the native functionality in Max to remove duplicate or static values in all keyframes. Also, though not supported by the Movetools format, patch objects will be converted to polygonal meshes on export. Additionally, during export Max has the additional feature of being able to freeze the scaling value of the objects and normalize them to 100 percent. This can be extremely useful when exporting between platforms, since the coordinate systems and scaling properties are seldom identical. Furthermore, the IK, Biped, and Look At controllers do not allow data to be read in. In order to use skeletons with these controllers, they must first be exported in the Movetools format, and then reimported into Max, where they will be assigned keyframable controllers. And finally, skinning is not currently supported, due to the way in which the skinning modifiers are implemented.

The Final Word

With the obvious advantages it brings to any production pipeline, Movetools is a must-have for any development team. With a few exceptions, notably the lack of skinning support in Max and the need for multiple machine-specific licenses, I was extremely pleased with this product and highly recommend it.

MOVETOOLS ★★★★

STATS

LAMBSOFT INC.
Minneapolis, Minn.
(612) 872-1700
www.lambsoft.com

PRICE
Permanent licenses are $1,800 per spoke; rentals are available for $300 per spoke per month.

SUPPORTED PACKAGES
Maya 2 or 2.5 (NT/IRIX), Softimage 3.8 (NT/IRIX), 3D Studio Max 2.5 or 3 (NT).

PROS

1. Plug-in interface requires no learning curve to use.
2. High-fidelity animation conversion with little or no data degradation.
3. Freshen option enables multiple animators and artists to work concurrently with the same content.

CONS

1. Multiple fees for multiple platforms: users need to purchase a licensed spoke for each platform.
2. Lack of portability: licenses are keyed to the machine and not the hardware lock, meaning artists have problems upgrading or switching machines.
3. Skins not supported in 3D Studio Max.

While Others Tune Their Engines, Our Customers Cross the Finish Line

Only one 3D game engine has what it takes to put you on the fast track from Day 1: NetImmerse from NDL. Here's why developers worldwide power their games with NetImmerse:

SPEED & CREATIVE FREEDOM

"With NetImmerse, we can concentrate on the design of the game itself, reducing production time." – *Mark Jacobs, Mythic Entertainment*

"NetImmerse allowed us to focus our energies on the content elements that set Panzer General 3D Assault apart from the competition." – *Dan Cermak, Mindscape*

CROSS-PLATFORM

Supports PC, PlayStation® 2, MacOS, and Linux.

VERSATILITY

Delivers superior 3D graphics for any style of game (action, RPG, Internet and others) from any perspective.

SOURCE CODE AVAILABILITY

"NetImmerse's source code gives us the flexibility to adapt our applications in a number of different ways." – *Ron Fortier, Digital Domain*

TRACK RECORD

More games are in development or have been completed using NetImmerse than with any other independent 3D game engine – by far.

SUPERIOR TECHNOLOGY

"We are taking advantage of a broad range of features, including skinning for natural-looking characters, and continuous level of detail, dynamic multiple-color lighting, and multitexturing to create rich environments." – *Jean-Noel Portugal, Dramaera*

EXPERT DOCUMENTATION, SUPPORT

"...the best documentation I have ever seen for a game-related SDK, by a wide margin." – *Jonathan Blow, Game Developer Magazine*

"Support is always prompt. Usually the person who wrote the software in question is available to provide answers." – *Bob Case, AniVision*

Contact NDL today to find out about the fastest path from 3D game conception to completion.



www.ndl.com
650-328-4388
sales@ndl.com

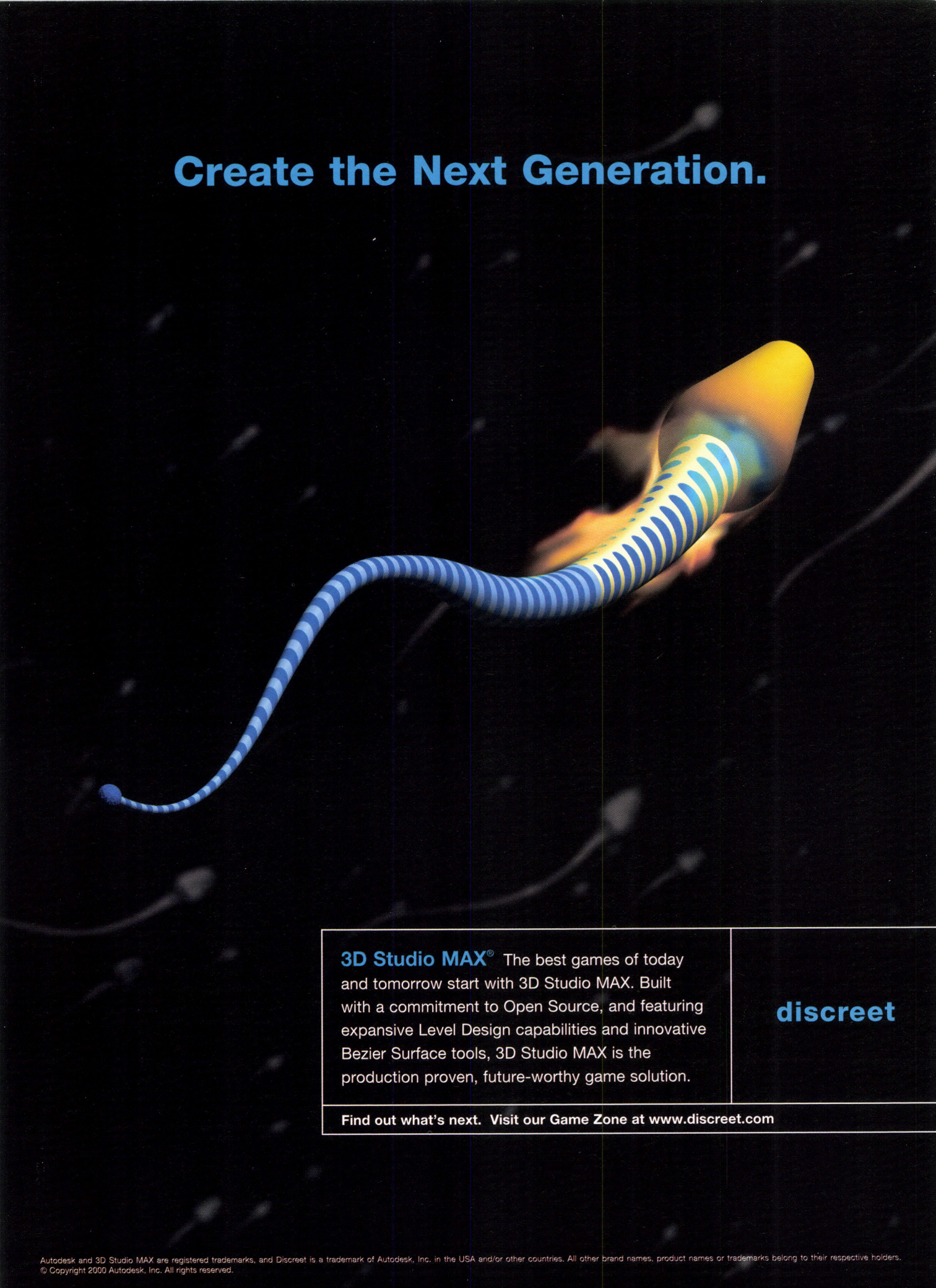


NETIMMERSE

Develop your game play, not your game engine.

Simon the Sorcerer 3D courtesy of Headfirst Productions
Prince of Persia 3D courtesy of Red Orb Entertainment
Panzer General 3D Assault courtesy of Mindscape SSI Division
Wild Wild West The Steel Assassin courtesy of SouthPeak Interactive
Wild Wild West™ ©1999 Warner Bros.
Racing car courtesy of AniVision, developer of *Net Race Live*

Create the Next Generation.



3D Studio MAX® The best games of today and tomorrow start with 3D Studio MAX. Built with a commitment to Open Source, and featuring expansive Level Design capabilities and innovative Bezier Surface tools, 3D Studio MAX is the production proven, future-worthy game solution.

discreet

Find out what's next. Visit our Game Zone at www.discreet.com

Autodesk and 3D Studio MAX are registered trademarks, and Discreet is a trademark of Autodesk, Inc. in the USA and/or other countries. All other brand names, product names or trademarks belong to their respective holders.
© Copyright 2000 Autodesk, Inc. All rights reserved.

In This Corner... The Crusher!

Squashing Stuff with Hardware-Friendly Animation Techniques

Everyone has their favorite memory of watching a classic Tex Avery or Chuck Jones cartoon. For many, these moments usually involve one of the characters being drastically deformed by a large, massive object. Simple 2D drawings made the physically impossible seem natural, believable, and fun for the kids.

Last month, I discussed how to use the power of new graphics hardware to give characters more life ("To Deceive Is to Enchant: Programmable Animation"). Using matrix deformation techniques along with interpolated morphing of meshes has gone a long way toward improving real-time character animation. However, creating the illusion of life needed for truly realistic characters requires more sophisticated techniques. According to John Lasseter (see References, p. 20), one of the chief advantages of computer animation is the ability to combine techniques in layers to achieve more complex and realistic results.

This idea of layering can be applied to real-time character animation to make characters more realistic. In past columns, I have talked about how a skeletal animation system is composed of a hierarchical structure of matrices (also referred to as bones) to provide the base animation layer for the character. The matrices are attached to a mesh "skin" by vertex weight assignments that relate each vertex to matrices in the system. These matrices are then kinematically animated to provide the motion.

However, fine details in a matrix deformation system are difficult to achieve. In order to create detailed articulation, such as for fingers or facial expressions, a great many matrices must be used. This increases both the production time required to create these characters and the processor time needed to render the character, thus reducing run-time performance.

Vertex morphing techniques are a very useful animation tool for efficiently achieving fine-detailed animation. Vertex morph animations for facial expressions and hand poses are easy to create and require minimal processing at run time. Typically, a single vertex morph target involves moving a very small subset of the vertices in a base mesh. The vertex morphing layer can provide the input to the skeletal animation layer, providing a very flexible animation system.

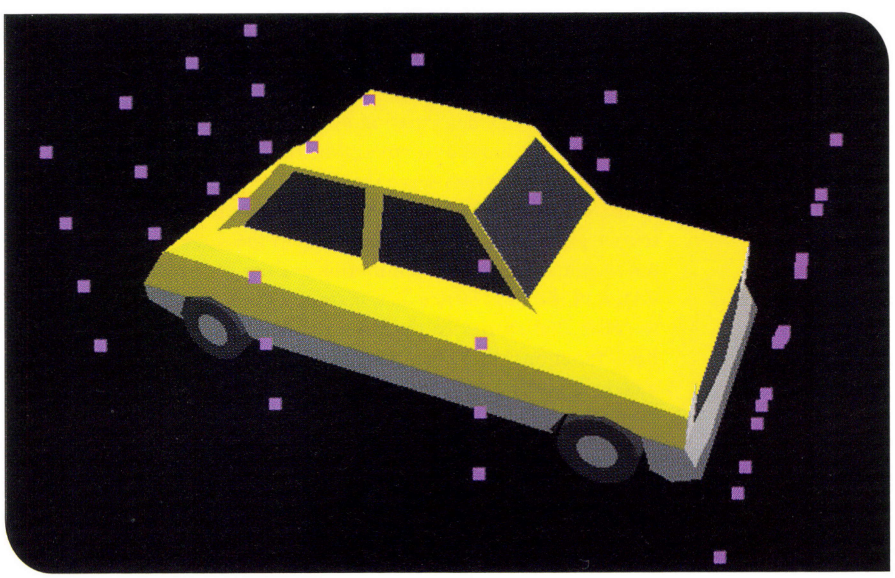


FIGURE 1. A free-form deformation lattice around a body.

AUTHOR'S BIO | *Cartoon inspirations come to Jeff after an afternoon at the brewery. If you have any visions you want to share, please contact him at jeffl@darwin3d.com.*

Dropping a Virtual Anvil on My Characters

After a long afternoon of Cartoon Network research, I decided that it was time to combine my cartoon renderer (see "Shades of Disney: Opaquing a 3D World," Graphic Content, March 2000) with some animation techniques so that I can start smashing things up. In their compelling work *The Illusion of Life* (see References), Frank Thomas and Ollie Johnston outlined the use of squash and stretch, exaggeration, follow-through, and overlapping action as key components for character animation. The combination of a skeletal animation system with vertex morphing described above supplies a lot of character control. However, these animation controls are not well suited to creating characters that can dynamically squash and stretch. As I described last month, the matrices in a skeletal animation system are full transformation matrices that can be translated, rotated, and scaled. These structures provide a great deal of local control over the vertices that the matrix influences. It certainly seems that matrix manipulation is a possibility for achieving some nice squishy effects. However, manipulating the control matrices individually can be tedious. Animators need a more intuitive parameterization of these properties in order to achieve fluid results.

Sederberg and Parry (see References) introduced the use of free-form deformations (FFDs) as an efficient method for animating soft bodies via a structural hyperpatch. By abstracting the control surface from the surface of the animated body, the deformation controls can be manipulated without regard to the model itself. This

technique has been used successfully to model semi-elastic surfaces.

An FFD works by positioning a 4×4 lattice of control vertices (CVs) around the model you wish to deform, as you can see in Figure 1. This lattice is aligned along the global X-, Y-, and Z-axes for clarity. It also makes sense to align it with the model's principal axes. These CVs are the controls the animator (or simulation, but I don't want to get ahead of myself) will manipulate to animate the object.

In order for the control vertices to change the model, I need to establish a relationship between the control lattice and the model. For this I'm going to use a cubic Bézier volume. This structure is composed of a 3D lattice of Bézier curves of degree 3 (cubic) which share control vertices. This gives me a total of 64 control vertices in a 4×4×4 grid. (For more on the mathematics of Bézier curves and patches, see Brian Sharp's *Game Developer* article on curved surfaces as well as Alex Ferrier's introduction to free-form deformations on Gamasutra.com, both in the References box.)

To evaluate a Bézier curve, I need a function that takes a point along the curve I wish to evaluate and returns the position. There is a function called the Bernstein basis function that serves this purpose. For a cubic curve, it takes the form

$$B^3(u) = (1-u)^3 p_0 + 3u(1-u)^2 p_1 + 3u^2(1-u)p_3 + u^3 p_4$$

where p_n represents the control vertices. To extend this function to a Bézier volume, it becomes a function of three variables (u,v,w)

LISTING 1. Converting the FFD function to vertex weights.

```
////////////////////////////////////////////////////////////////////
// Function:        SetFFDWeights
// Purpose:         Approximate an FFD by setting up control
weights
// Arguments:       Pointer to base mesh visual structure
////////////////////////////////////////////////////////////////////
void SetFFDWeights(t_ToonVisual *visual)
{
/// Local Variables ///////////////////////////////////////////////
   tVector *vertex;
   int loop,cvLoop;
   float  XBasis[4], YBasis[4], ZBasis[4];
   float  u, v, w;
   float  *vertexWeight;
   int    px,py,pz;
////////////////////////////////////////////////////////////////////
   // Allocate the space for all the weights
   visual->weightData = (float *)malloc(visual->vertexCnt *
      FFD_NODE_COUNT * sizeof(float));
   vertex = visual->vertex;
   // Go through all the vertices
   for (loop = 0; loop < visual->vertexCnt; loop++, vertex++)
   {
      // Find where each vertex is within the FFD grid
      // Effectively scales each vertex to 0-1
      u = (vertex->x - g_FFDmin.x)/(g_FFDmax.x - g_FFDmin.x);
      v = (vertex->y - g_FFDmin.y)/(g_FFDmax.y - g_FFDmin.y);
      w = (vertex->z - g_FFDmin.z)/(g_FFDmax.z - g_FFDmin.z);
```

continued on page 18

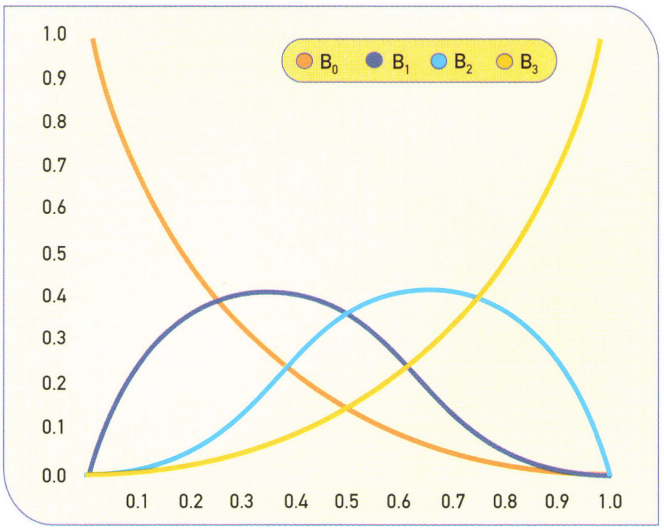


FIGURE 2. Bézier basis functions.

which represents the 3D position within the Bézier volume. The full formula for the Bézier volume basis function is:

$$B^3(u,v,w) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} p_{ijk} B_i^3(u) B_j^3(v) B_k^3(w)$$

It seemed to me that layering the FFD technique into my animation system would allow me to get the results I wanted. However, in order to deform a mesh using this FFD lattice, I need to pass every vertex through this function to evaluate its deformed position. That's 64 evaluations of the Bernstein basis function at every vertex. This amounts to quite a few calculations that would fall under the CPU's responsibility. It occurred to me that the control vertices look an awful lot like matrices in my standard animation pipeline. Consumer graphics hardware that accelerates matrix deformation techniques is becoming standard and will be widely available by end of this year. If I can frame the FFD problem in terms of a matrix deformation system, I can use this hardware to relieve the CPU and also streamline my animation pipeline.

The basis functions serve the role of relating the control vertices to the vertices in the base mesh. For matrix deformation, the vertex weights relate the control matrix in the same way as the vertex weights in a skeletal animation system. If I treat each control vertex in the FFD lattice as a matrix, I need to create the weights to relate that matrix to the base mesh. This brings me back to the basis functions. Each control vertex in the control curve influences a certain portion along the length of the curve. If I examine the influence each CV has on the curve, I get the formulas:

$$B^3(u) = (1-u)^3 p_0 + 3u(1-u)^2 p_1 + 3u^2(1-u)p_3 + u^3 p_4$$

$$B^3(u) = B_0^3 p_0 + B_1^3 p_1 + B_2^3 p_3 + B_3^3 p_4$$

$$B_0^3 = (1-u)^3$$

$$B_1^3 = 3u(1-u)^2$$

$$B_2^3 = 3u^2(1-u)$$

$$B_3^3 = u^3$$



artists **animators** **game developers**

When your reputation depends on your performance

and reliablity

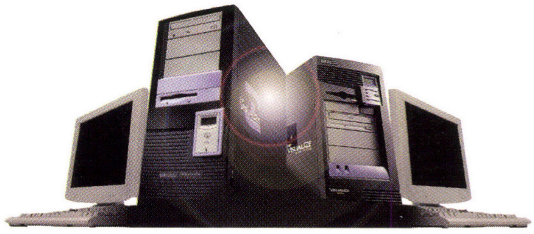
and graphics power

and interactivity

– our performance is your art.

hp®

invent



HP VISUALIZE WORKSTATIONS

visit: www.hp.com/info/dcc3

intel inside

pentium® III

Your reputation is built on your 3D animation, visual effects or games. And HP performance and reliability are exactly what you need because an HP VISUALIZE workstation is body armor for your reputation.

HP's VISUALIZE *fx*⁺ graphics deliver the extreme performance you need to work interactively with million+ polygon models. Great graphics, the latest Intel® Pentium® III processors, HP-engineered reliability, and at a price you can afford.
Our performance is your art.

Intel, the Intel Inside logo, and Pentium are registered trademarks of the Intel Corporation.
Images (from left to right) courtesy of Alias|Wavefront, Major Damage™, ©2000 Chris Bailey and EPI/Ron Dicker.

LISTING 1 (continued). Converting the FFD function to vertex weights.

continued from page 16

```
        // X Bezier Basis Functions
        XBasis[0] = (1.0f - u) * (1.0f - u) * (1.0f - u);
        XBasis[1] = 3.0f * u * (1.0f - u) * (1.0f - u);
        XBasis[2] = 3.0f * u * u * (1.0f - u);
        XBasis[3] = u * u * u;

        // Y Bezier Basis Functions
        YBasis[0] = (1.0f - v) * (1.0f - v) * (1.0f - v);
        YBasis[1] = 3.0f * v * (1.0f - v) * (1.0f - v);
        YBasis[2] = 3.0f * v * v * (1.0f - v);
        YBasis[3] = v * v * v;

        // Z Bezier Basis Functions
        ZBasis[0] = (1.0f - w) * (1.0f - w) * (1.0f - w);
        ZBasis[1] = 3.0f * w * (1.0f - w) * (1.0f - w);
        ZBasis[2] = 3.0f * w * w * (1.0f - w);
        ZBasis[3] = w * w * w;

        // Pointer to Place to store weight data
        vertexWeight = &visual->weightData[loop * 64];
        // Go through the control vertices
        for (cvLoop = 0; cvLoop < FFD_NODE_COUNT;
           cvLoop++,vertexWeight++)
        {
           // Some quick math to find the component indices
           px = FFD_WIDTH - (cvLoop % FFD_WIDTH) - 1;
           py = FFD_HEIGHT - (cvLoop / (FFD_WIDTH * FFD_HEIGHT)) - 1;
           pz = FFD_DEPTH - ((cvLoop % (FFD_WIDTH * FFD_HEIGHT)) /
              FFD_WIDTH) - 1;

           // set the vertex weight for this CV
           *vertexWeight = (XBasis[px] * YBasis[py] * ZBasis[pz]);
        }
     }
  }
  /// SetFFDWeights ////////////////////////////////////////////////
```

These formulas show the influence each control vertex has on the curve, which you can see graphically in Figure 2. Each control vertex has an influence over a region of the volume. These functions will provide the vertex weighting data I need.

To attach a FFD lattice to an object, I take the base mesh and place it within the FFD lattice. The weights are calculated for each vertex by scaling the vertex position to a value between 0 and 1. This represents the relative position of the vertex within the lattice. The scaled value is plugged into the three basis functions for each control vertex and out pops a vertex weight that relates the mesh vertex to that CV. Listing 1 contains the code that calculates these weights. You can also see the influence of a single CV on the mesh in Figure 3.

One restriction required in order for matrix deformation to work is that the sum of the weights on any one vertex must equal one. Fortunately for us, the wonders of mathematics are working

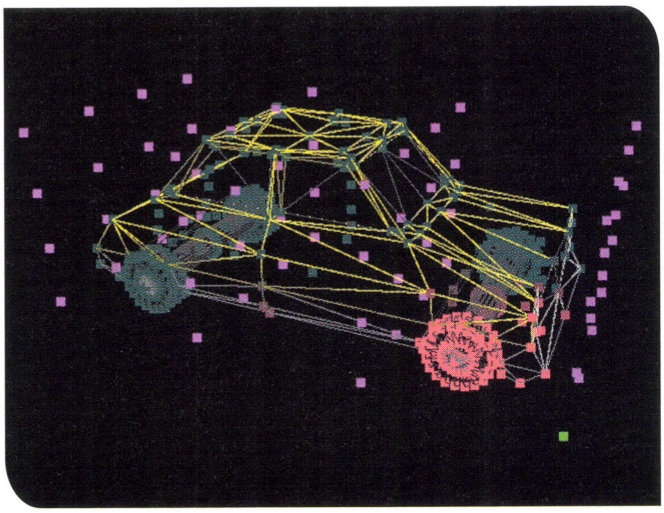


FIGURE 3. Weight values for a control vertex.

in our favor. Due to the very nature of the Bézier basis functions, the sum of the influences at any point along the curve is always equal to one. You gotta love how well that works out.

Once these weight values are calculated for each vertex, I can run this object through my matrix deformation system and start moving control vertices around. As each CV moves, it deforms the base mesh through the weight values. In fact, I get a bonus over the traditional FFD system. I can apply other transformations on these control points. I can rotate and scale them, giving me even more control over the mesh. However, I still need to move each CV individually to make anything happen. I will need to add a control mechanism to make them move together.

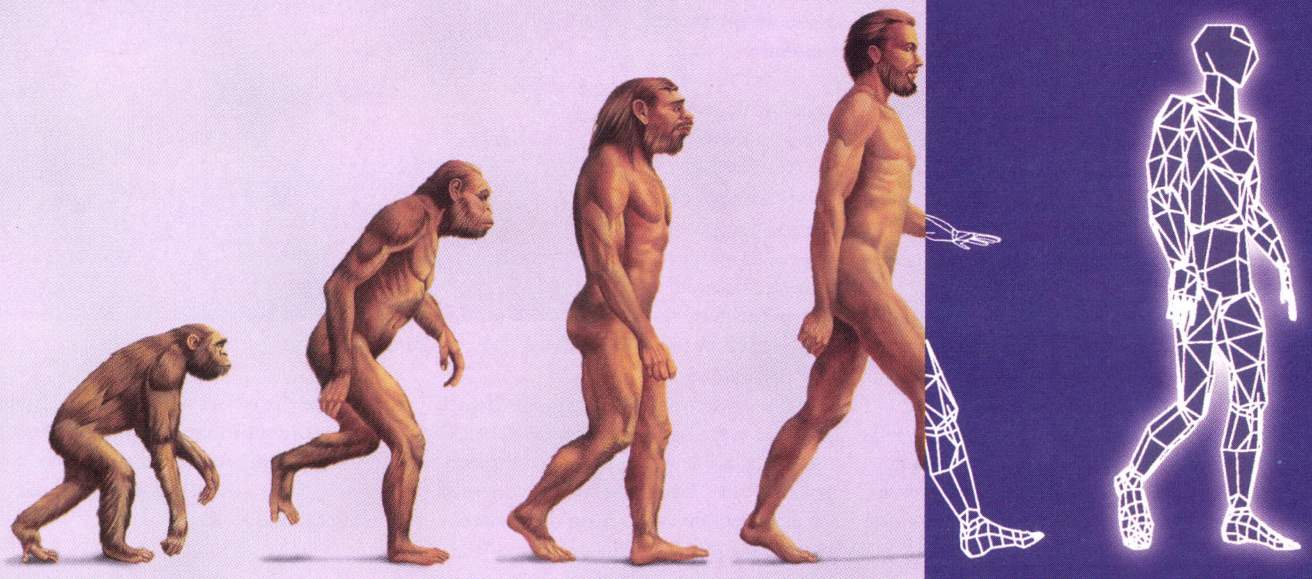
Controlling the Squishy Beast

For many applications, manually positioning the FFD control vertices will work fine. Often, though, I will want them all to move together like a single flexible object. Fortunately, I have played around with something like this in the past. You may recall my column last year on the topic of soft-body dynamics ("Collision Response: Bouncy, Trouncy, Fun," Graphic Content, March 1999). In that column, I connected point masses together using dampened springs. I could then toss those objects around and they bounced off the walls and floor in a flexible manner.

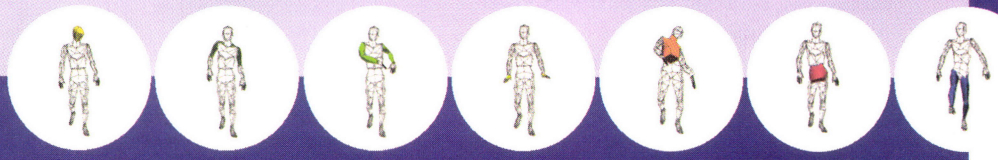
For this application, I will make the point masses the control vertices in my FFD lattice. I then connect those points together with a network of springs in the same way as I did in my March 1999 column so the lattice will be somewhat stable when I drop it. When I run the object through my particle dynamics simulator, the control vertices start bouncing around. Since these control vertices are used to deform the base mesh, the mesh bounces along also. Just for fun, I applied the cartoon shader to the objects so I could really get that Saturday morning, "bang him on the head with a skillet" feel.

You can see the application in action in Figure 4. I loaded in my cartoon car and bashed around some of the CVs to flatten the roof.

MOTION CAPTURE ANIMATION TAKES AN EVOLUTIONARY STEP FORWARD.



Sophisticated motion capture for every production budget has arrived. MOTEK introduces UNICA,™ the web-based animation solution that lets you rapidly custom-build and blend data sets from the world's largest motion capture library, view a preview, then purchase and download — at one-tenth the cost of original production. With UNICA, the world of motion capture is at everyone's fingertips. Come play on our site right now and watch your imagination take shape right in front of your eyes.



WWW.E-MOTEK.COM

603-641-1300
INFO@E-MOTEK.COM

Motek™
Technology in Motion

© 2000. MOTEK Motion Technology Inc. MOTEK, the MOTEK logo, and UNICA are trademarks of MOTEK Motion Technology Inc.

Putting It All Together

Adding this FFD lattice technique to a character animation system opens up some interesting possibilities. The FFD can be positioned in the skeletal hierarchy such that transformations are inherited from parent matrices. That way an FFD lattice can be applied, for example, to an upper arm so that the muscles will bulge. You would need to be careful with how the weights blend across the FFD and skeletal links. However, I have found that scaling and blending weights works very well.

There are a few problems with the use of the mass-and-spring system for FFD animation. My current system does not preserve the volume of the original control mesh. What that means is that the spring system can find valid configurations where it has collapsed inside itself. This may not be totally realistic for certain solid but flexible objects. For an object like my car, though, it works in my favor. I sometimes want the object to collapse inside itself and stay there. At the GDC in March, Alexis Mather of Matrox Graphics showed how hardware-accelerated matrix techniques can be used in this way to simulate damage on a complex car model. If such behavior were not desirable, however, I could add a lot more springs to the control mesh or I could use another method for dynamically connecting the control points that preserves the volume of the object. I may take a look at that issue in another column.

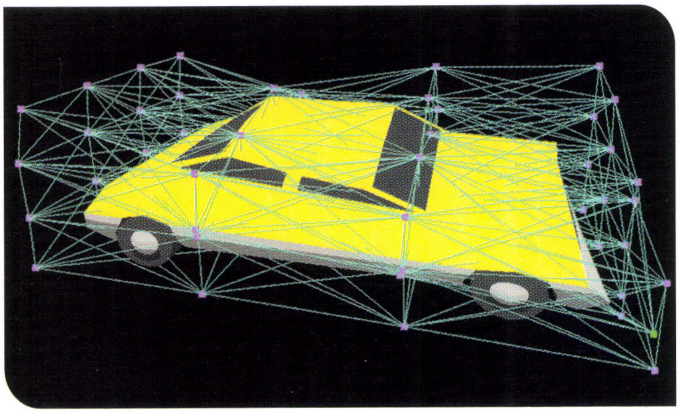


FIGURE 4. A taxicab in a meteor storm.

Another problem involves rendering. When I deform the mesh, I am moving vertices all around. This changes the surface quite a bit and I am not currently adjusting the surface normals to match, which causes some problems with the shading model. Unfortunately, this is a tricky problem. To fix it, I would need to rebuild the vertex normals by creating new face normals and averaging them to get new vertex normals. This is processor-intense, but not terribly hard to code up. I leave that up to industrious readers to add to the sample application.

As a second layer of abstraction, it would be interesting to make the FFD lattice deform a skeleton inside the object instead of individual vertices in the mesh. That would definitely speed up the calculations, as there would be a lot fewer points to process. However, it would also lower the amount of control.

There's also nothing stopping you from using lower- or higher-degree Bézier volumes. I chose cubic because it seemed to provide a good flexibility-to-performance tradeoff. For other objects, more or less control may be needed.

Another interesting extension to this technique would be to make some of the lattice springs active instead of passive, thereby creating virtual "muscles" that animate the object automatically. I will look more into that next month. For now, grab the application and source code at www.gdmag.com and start smashing things around.

CLEARING OUT MY E-MAIL BOX

One of the fun things about writing this column is all the great mail I get from readers. For anything I wonder about or miss, one of you always is quick to let me know about it.

In last December's column on 2D water effects ("A Clean Start: Washing Away the Millennium"), I mentioned that I didn't know who wrote the original version of this idea. Several people wrote in, the first being Juan Carlos Arevalo Baeza who wrote that the effect was done for Heartquake, a demo entered in a competition in Helsinki by Arturo Ramirez-Montesinos of the demo group Iguana in 1994. He based the idea on a crude approximation of the 2D general wave propagation formula. You can get the original demo at ftp://x2ftp.oulu.fi/pub/msdos/programming/iguana/heartq.zip.

I also received a note from Fabio Policarpo who is working on a book on game programming with Alan Watt. He integrated the cartoon rendering technique into his Fly engine that will be the basis of the book. He has a great demo of a cartoon car driving around a terrain. You can download the demo at www.paralelo.com.br/download/car-toon.zip.

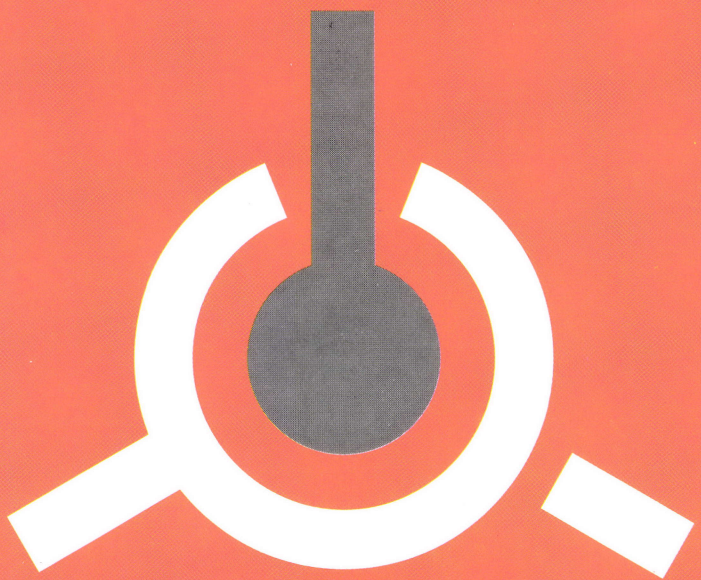
ACKNOWLEDGEMENTS

Thanks to Alexis Mather and Jason Della Rocca of Matrox, Alex Ferrier, Andrew Bond of Havok.com, and Casey Muratori of RAD Game Tools for discussing this issue with me.

REFERENCES

- Lasseter, John. "Principles of Traditional Animation Applied to 3D Computer Animation." *Proceedings of Siggraph '87*. In *Computer Graphics* (Vol. 21, No. 4): July 1987.
- Thomas, Frank, and Ollie Johnston. *Disney Animation: The Illusion of Life*. New York: Abbeville Press, 1984.
- Sederberg, Thomas, and S. R. Parry, "Free Form Deformations of Solid Geometric Models." *Proceedings of Siggraph '86*. In *Computer Graphics* (Vol. 20, No. 4): August 1986.
- Sharp, Brian. "Implementing Curved Surface Geometry" (*Game Developer*, June 1999).
- Ferrier, Alex. "Real-Time Soft-Object Animation Using Free-Form Deformation." www.gamasutra.com/features/19990827/deformation_01.htm.
- Chadwick, John, and others. "Layered Construction for Deformable Animated Characters." *Proceedings of Siggraph '89*. In *Computer Graphics* (Vol. 23, No. 3): August 1989.
- Mather, Alexis. "Content Creation for Hardware Accelerated Skinning." Game Developers Conference 2000. www.matrox.com/mga/dev_relations or devel@matrox.com.

The Data Management Solution for Game Developers



alienbrain®

As a comprehensive solution alienbrain® provides a data management environment that keeps track of your content from creation to delivery. Version control, file sharing, annotations, searching, process automation and status tracking are just some of the features that allow your team to be as productive at the size of 20 as they were with 5, and work with 100,000 files as well as they did with 5,000.

genius | intelligence | control | knowledge

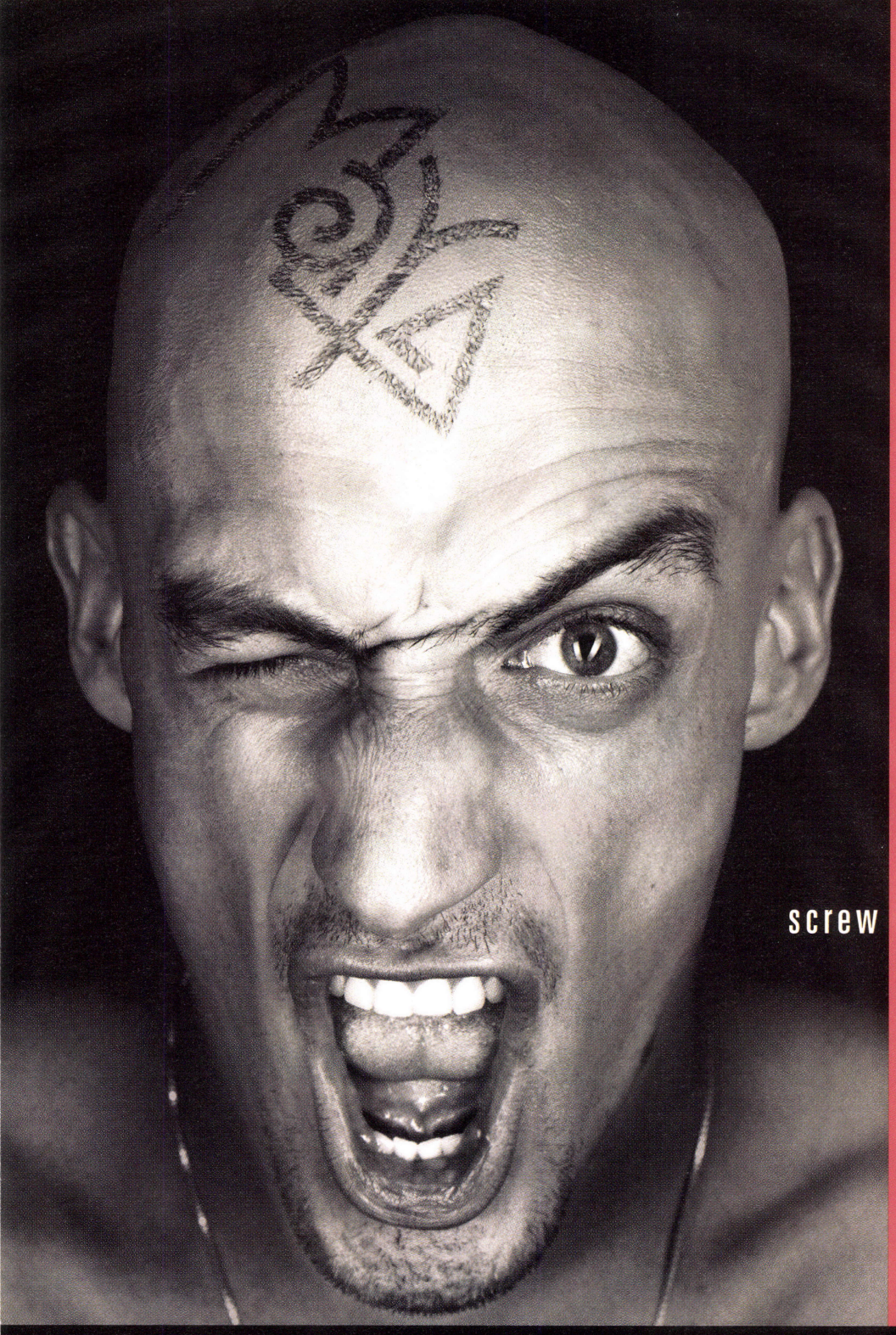
Game Developer front line AWARD

http://www.nxn.co.uk
http://www.alienbrain.com
e-mail: info@nxn.co.uk

NxN and alienbrain are registered trademarks

"See NxN at E3, Booth 868, South Hall"

NXN



work hard. play hard.

according to that statement,

it's all about balance.

why?

unbalanced people are

way more interesting.

true geniuses forget to put

their pants on in the morning.

screw balance!

find stuff that will make work easier,

so you can play harder.

Alias | wavefront

MAYA™ CAN YOU IMAGINE

North America	800.447.2542
Europe	800.4125.4125
Asia-Pacific	81.3.3470.8282
Latin America	525.203.0030

Starting at $2995 USD. Available on Windows NT and IRIX.
To find out more about the most powerful 3D gaming tools money can buy, visit www.aliaswavefront.com

Come visit our E3 Demo Suite at 501 C West Hall
Come visit us at the 3D Conference and Expo

"The Best Laid Plans..."

Repenting Our Art Sins

It's been estimated that for every PC and console game that successfully makes it onto retail shelves, there are between three and five titles which get axed at some point in their development. And of the thousands of games that do make it to market, only a chosen few ever reach the triple-A status that separates a mediocre title from one of greatness. In an ideal world, the development community would learn from its mistakes and constantly upgrade its performance so that the games we make would get better and better. This is clearly not the case however, and the number of graphically mediocre and bug-ridden games released continues to mount each year.

It seems clear then, that a lot of misdirected effort is being spent on creating product that either never sees the light of day, or does so only as a stopping point on the fast track to the bargain bin. Why does it have to be this way? Shouldn't we, as developers, be learning from our past mistakes? Or are we wrapped up in some kind of adolescent invulnerability complex which makes us immune to the advice and experience of others? If the Postmortem columns in this magazine are any indication, it seems clear that we are continuing to beat our heads against the wall.

This month, I'm going to depart from the standard tips and tricks format for art generation and look at content-creation problems at a more fundamental level. I'll identify three of the major areas where developers tend to misstep, discussing both the methods that work and the mistakes that keep coming back to haunt us. And hopefully, if the planets are aligned and the moon is full, a few of us will actually remember and adhere to the lessons learned by ourselves and our peers.



FIGURE 1. Clockwise from top left, screenshots from ABE'S ODDYSEE PS2, QUAKE 3: ARENA, NOCTURNE, and TEKKEN 3. In each of these titles, the art serves a different purpose.

AUTHOR'S BIO | *Mel Guymon has been animating in the gaming industry for several years. When he's not at his desk pushing polygons, he can usually be found at the local Barnes and Noble, slumming for reference materials. Mel can be reached at mel@infinexus.com.*

Creation with Purpose

Before we begin analyzing the potential problems associated with in-game art generation, it's crucial for us to realize that the art we generate is far more than just eye candy for the player, and as such, it deserves the same level of attention to detail as the coding and design aspects of a game. In addition to being the first thing that a player usually notices about the game, the art can serve as the vehicle through which the storyline is conveyed, either through interactive cutscenes, or through the use of a consistent, cohesive art style. Alternately, as in an action/adventure game, the art can serve as the supporting canvas on which the game play takes place. Or finally, as in the case of a first-person shooter, the art need only be capable of maintaining the immersive quality of the game-play space. (See Figure 1 for some examples of different art functions in various games and genres.)

In each of these cases, the art serves a specific role within the game engine. In order for an artist to be successful, he or she must first understand and embrace the role the art will play within the game. This will help to keep the team as a whole focused on the important aspects of art generation and help minimize the time spent on extraneous or inappropriate content. Once the team has a clear understanding of what role the art will serve in the game, the images in the art bible can be placed in context.

The Art Bible: Working from a Blueprint

One of the most common complaints in the "what went wrong" section of the Postmortem column is that the art in the game had no coherence, no vision. In most cases, this is due to a lack of artistic direction in the form of a readily accessible and updated style guide or "art bible." Ideally, the art bible contains both reference photographs and artist-generated images depicting every scene in the game. It should contain as much detail as possible about every object in the world, down to the color and surface properties of the textures. While the art bible should be started at the same time the R&D phase for the game begins (or at least in concert with the initial resource production), the document must be treated as a living organism, being updated and kept current as the project evolves. For artists, this document should be the blueprint for the world they create.

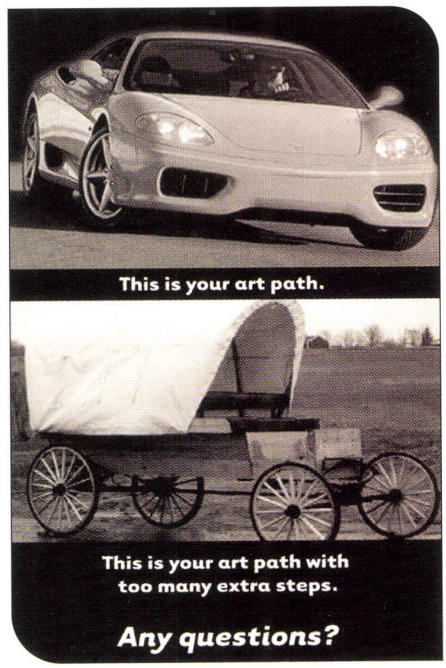
Think of a contracting firm constructing a skyscraper. It would be insane to expect them to build it without a blueprint, and even if they tried the results would be disastrous. So too are there serious ramifications for attempting to create art without an artistic style guide. Either the vision will be incomplete or nonexistent, or poorly executed and underdeveloped. Furthermore, once the construction of a skyscraper has commenced, changes to the blueprint may have repercussions which are hard to rectify. In the same manner, making changes to the art bible should not be done in a haphazard way, particularly when they affect objects which have already been built.

The most common complaint I hear about creating an art bible is, "I know what a tree looks like, so why do I have to put a picture of a tree in the art bible?" The answer to this question — and any of the many other "why" questions — is simple. First, for most objects, sketching out or finding photo resource for an object takes much less time than actually building it in 3D. So it follows that the most efficient way to iterate on the look is on paper, not in 3D space. And second, the art bible, when taken as a whole, is the best place to see the coherent artistic vision before it ever gets built. When something doesn't quite fit the vision, it can be identified before a single polygon is constructed.

The Art Path: Opening the Floodgates of Creativity

Another common thread in the Postmortem saga is the lack of an efficient, GUI-based protocol for converting and editing art resources in the game engine. For art leads and art directors, few things are more important than creating an efficient and effective art path for getting resources into the game. The art path is the main artery through which all resources will enter the game engine. The primary test of an efficient and effective art path is how long it takes for an artist or designer to load and play-test art resources within the game engine. With a well-crafted art pipeline, artists and designers should be able to preview and game-test their work within a few minutes of creating it. A poorly executed art path wastes artists' and designers' time as they try to merge resources with the game engine. This can cause potentially fatal problems at several levels. The more time lost in converting and managing data, the less time there is for creativity and innovation. Consider further that the more tedious it is to get resources into the game engine, the longer the iterative loop for resource debugging and play-testing. This can result in a lack of polish in both look and game play. If it takes two hours to edit and preview an object in the world, how motivated will the artists be to fix a single erroneous mapping coordinate?

Here's a real-life example that describes this problem with stunning clarity (the names have been changed to protect the innocent). Bob is art-directing a real-time 3D action/adventure game that takes place in a large exterior terrain system. The art path that has been implemented starts with 3D Studio Max, but the data must pass through multiple external formats and programs to gain lighting information and be converted into a data format readable to the game engine. Furthermore, the massive levels must be "BSP'd" (that is, have their BSP trees calculated) in order to preview any changes made to the levels. As Bob comes into the office to check up on his artists and level designers, he notices that



most are sitting idle at their desks. When he asks them what they are waiting on, one of the modelers chimes in to tell him that he had made a series of texture-mapping adjustments to the level, and he was just waiting for the engine to finish calculating the BSP tree for the level so he could preview the changes he made in the game engine. The problem is that it took more than five hours to BSP the level. How much more motivated and creative would the artist have been if he could have previewed his work in five minutes rather than five hours? Rather than interactively adding value to the game play and aesthetic of the level, the developers in this situation were forced to waste time waiting on an inefficient process.

Getting the Most Bang for Your Buck

The third most common problem with art generation is the lack of focus on what really matters. All of the developers should have a clear idea of what the purpose of the art is within the game. If what you're doing doesn't serve a specific purpose and add value to the product, something is wrong. Adding in a special effect or technical nicety which only another developer would appreciate is generally not a good idea. This is not to say that innova-

dynamics toolkit 2.0

If you've looked at using dynamics middleware before, you'll find our latest offering faster, more stable and powerful enough to really let you make a difference to your game.

If you havn't considered it before, we'd like to show you what we can do.
You'll find our Dynamics Toolkit is designed to mesh with your game design not dictate it.

We can give you a physics solution that works out of the box or can be tailor made for your title. Most components are supplied with full source code and open pricing and all are available on PS2 and PC
(Windows or Linux).

Be one of the first to see it in action at
dev.mathengine.com

to see our all new technology

register now at dev.mathengine.com

tion and uniqueness aren't good qualities to have as an artist, quite the contrary. However, when deciding whether to include a certain feature, it's important to remain objectively critical about the value of the feature. What actual value — value perceptible to the player — does it add? How does it positively or negatively impact the game-play experience? Was it created using a known technique that others on the development team will be able to grasp? What is the cost in man-hours to implement the feature? And how will it affect the run-time experience on the low-end target platform? Features that should undergo such scrutiny include things as specific as the number of polygons in a character's head or as broad as the decision whether or not to use global vertex lighting instead of lightmaps.

The ultimate litmus test of a new feature, however, is what portion of the product's purchase price is the user willing to allocate for the feature. For example, if in your action/adventure game you add the ability to use inverse kinematics to keep your character's feet on the ground and it costs $300,000 of your $2 million budget, essentially that means players will spend $6 of their $40 purchase price on that feature. Put yourself in the player's shoes and ask where you'd like that extra $300,000 to be spent. Most likely, the subtle gimmick which impresses the other developers is totally wasted on the player, who'd much rather see better AI, a fully developed story, and a richer, more diverse world.

Wrap Up

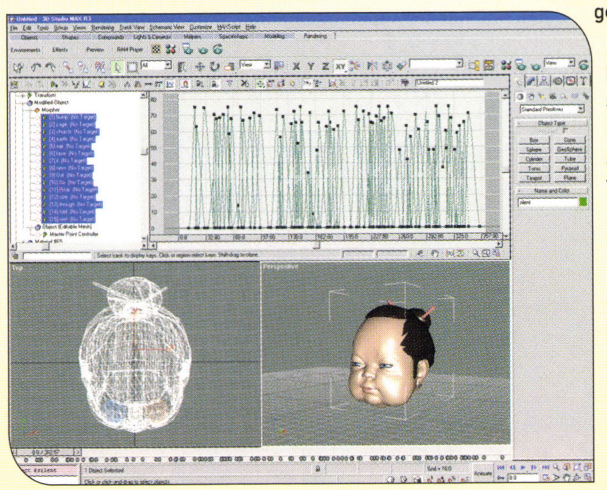
In the game development industry, the road to innovation is paved with the remains of projects which hovered near greatness but were doomed to ignominy before they ever hit the shelves. While the cause of these misfortunes can sometimes be blamed on an overeager publisher or a less-than-receptive market, often a game's demise results from developers' lack of discipline and attention to detail, ideals which were cast aside in the name of creative freedom. The ongoing rash of buggy and visually substandard titles serves as testament to the unspoken yet pervasive notion that process and methodology have no place in game development. Ultimately it will be the teams that have both the vision to be innovative and the discipline to adhere to process that will win the respect of players and publishers alike.

(Disclaimer: The author of this column has never made any of the mistakes referenced in this piece. In particular, he has never implemented a heinously complex art path or neglected the construction of an art bible, or added fallacious and needlessly complicated special effects, and he has never, ever, *ever* succumbed to creeping featuritis. And anyone who swears they saw him commit such acts was probably under the influence of a mind-altering drug, or it was my evil twin brother Mort, take your pick.)

PLUG-IN POWER!

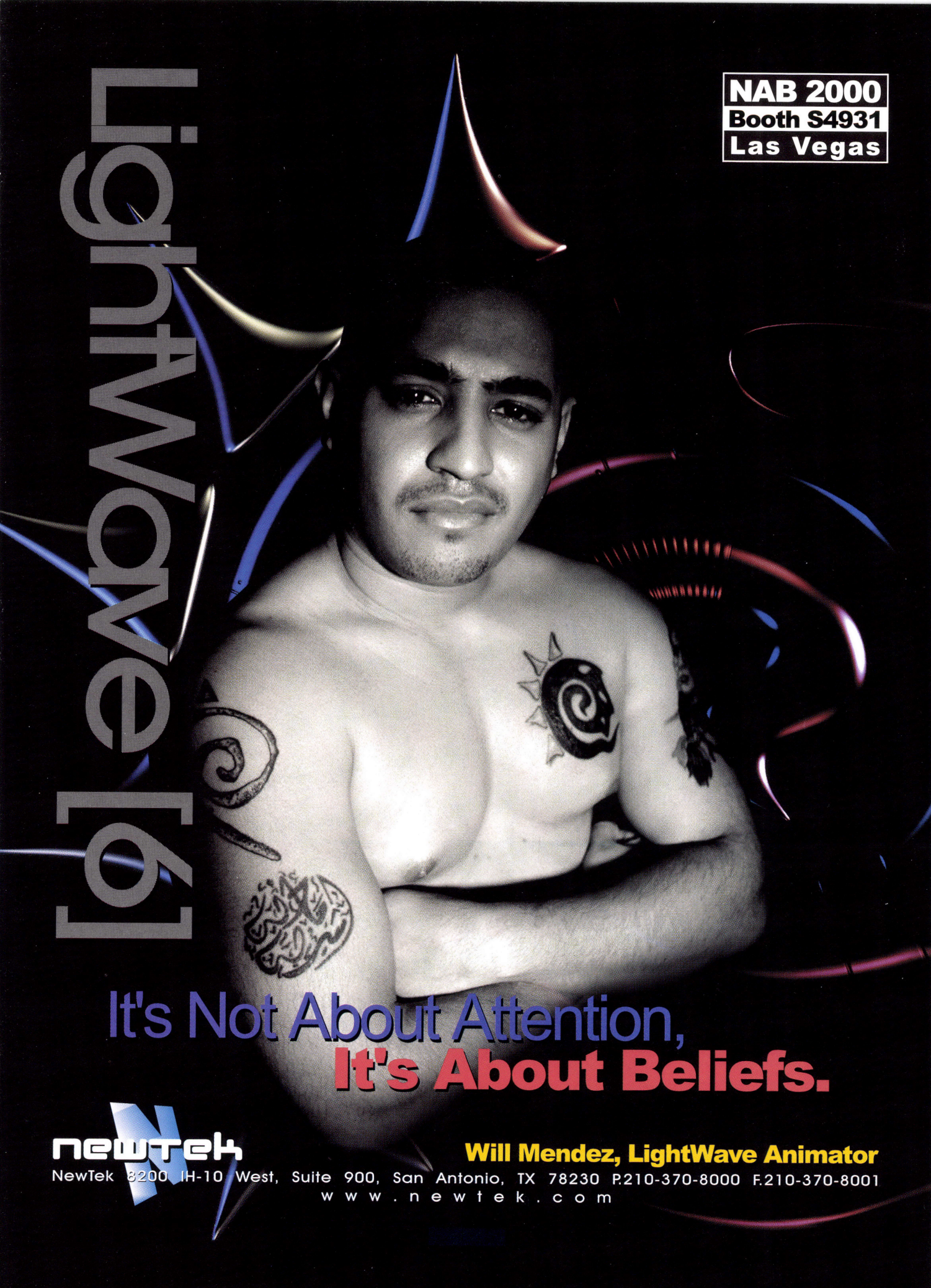
LIPSINC'S VENTRILOQUIST

A tool like Lipsinc's Ventriloquist has long been on the wish lists of animators, for whom the extremely dreary process of lip-synching has become more than a painful chore. One of a suite of tools for 3D Studio Max, Ventriloquist automates this tedious process by analyzing a sound file and automatically identifying the phonemes in the audio track. (Phonemes are the building blocks that form the sound patterns we associate with speech. Depending on which method you ascribe to, there can be anywhere from nine to 40 different phonemes in English. The visual equivalent of a phoneme is a viseme, which is the shape your mouth and tongue form when pronouncing the phoneme.) Though still dependent on a good animator to create the actual visemes, this product eradicates the most difficult step in the process, that of animating a character's mouth in sync with the spoken word.

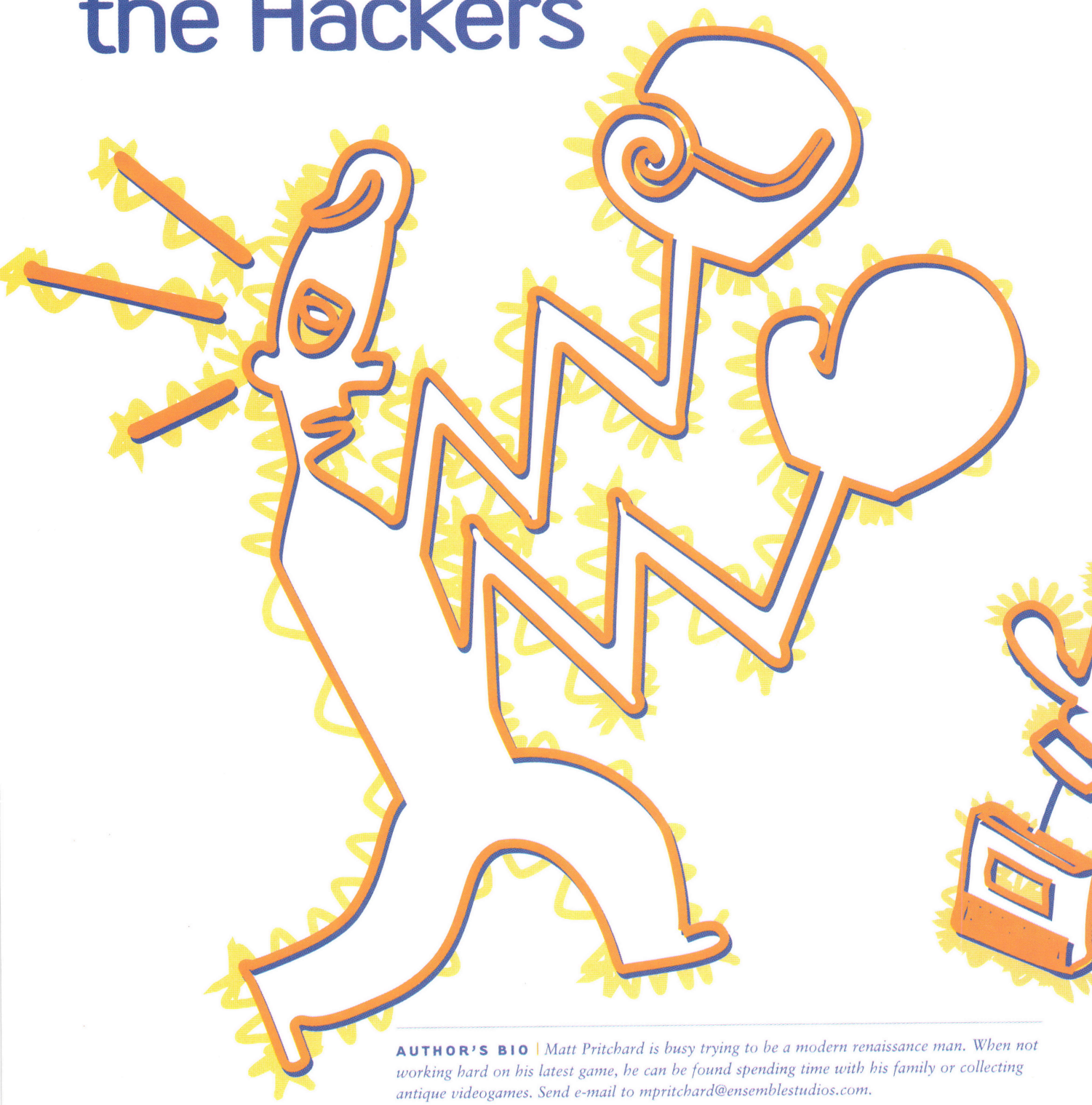


The process is very straightforward. As in standard lip-synching, the animator creates and assigns a set of morph targets, corresponding to the desired list of visemes to be used. Ventriloquist has a preset list of 16 visemes which are created and then linked to controllers in the plug-in. Once the visemes are created and assigned, the animator's lip-synching work is mostly complete. Any number of audio tracks can then be loaded in, and the plug-in automatically analyzes the sound file for phonemes and assigns the correct animation, freeing up the animator to focus on secondary head and eye movement. The screenshot at left shows the animation tracks that were generated with Ventriloquist.

And while Ventriloquist is strictly a production tool for use with canned sequences and prerendered scenes, Lipsinc is about to make their SDK available in two new products called Talk Back and Talk Now, which are able to accomplish the same thing in real time. According to Scott Curtis at Lipsinc, the Talk Now SDK, which can sample voice tracks coming in from a multiplayer game or in an audio-based chat program, is able to generate pretty good results, with only a three- to 500-millisecond lag time. Clearly this is something that game players and online developers are salivating for, and I'll be eager to see it when it ships. Ventriloquist stands out as a production-enabling weapon which will significantly help raise the bar of RT3D entertainment and is a definite "must-have" for anyone working with lip-synched characters.


LightWave [6]
NAB 2000
Booth S4931
Las Vegas
It's Not About Attention,
It's About Beliefs.
newtek
Will Mendez, LightWave Animator
NewTek 8200 IH-10 West, Suite 900, San Antonio, TX 78230 P.210-370-8000 F.210-370-8001
www.newtek.com

How to Hurt the Hackers



AUTHOR'S BIO | *Matt Pritchard is busy trying to be a modern renaissance man. When not working hard on his latest game, he can be found spending time with his family or collecting antique videogames. Send e-mail to mpritchard@ensemblestudios.com.*

The Inside Scoop on Internet Cheating and How You Can Combat It



I had planned to begin this article by sharing my own true experiences with online cheating as it pertained to a particular game. But I think the long version of my story would cast an unnecessarily negative light on the game and the company that made it. And since the developers are good friends of ours, I'll stick to the short version that goes like this.

Last year I became hooked on a certain first-person shooter (FPS) game. After a couple months of addictive online gaming, I became convinced that some players were cheating and things suddenly changed that day. I was ready to walk away from the game in disgust and tell everyone else to do the same. Instead, I decided it was time to learn what I could about the alleged cheaters, their motivations, and most importantly their methods. In my case, I discovered at least three distinctly different methods of cheating that could explain what I experienced — though as just a player I could not prove conclusively which methods, if any, were being used against me.

The aim of this article is to bring the subject of online/multiplayer cheating out of the shadows and talk about it in terms of real problems with real games and to help build a framework for classifying and understanding the various details. I will cover some of the ways that players are able to cheat at various games; at times I will go into the working details, ways to prevent those cheats, and limitations of various game architectures as they relate to multiplayer cheating. This is by no means a comprehensive and exhaustive tome on the issue, but it is a start. There is a serious lack of information on this subject, and paranoia among developers that talking about it will reveal secrets that will only make the problem significantly worse. Several individuals at various companies declined to talk to me about cheating and their games for this and other similar reasons. I respect that, but I think developers have everything to gain by sharing our knowledge about cheaters and how to combat them.

Just how seriously should you as a developer take the possibility of online cheating? If your game is single-player only, then you have nothing to worry about. But if your game is multiplayer only, the success of your entire product is at stake. If your game does both, you're somewhere in the middle. As more games are released with online play as an integral component, drawing ever-larger audiences (and the corollary development of online communities and sites based around the game), it becomes ever more important to insure that each online game player experiences what they believe to be a fair and honest experience. I'm reminded of a quote from Greg Costikyan's excellent report, "The Future of Online Gaming" (www.costik.com): "An online game's success or failure is largely determined by how the players are treated. In other words, the customer experience — in this case, the player experience — is the key driver of online success." Our short version is, "Cheating undermines success."

Consider the well-known case of Blizzard's DIABLO — deservedly a runaway best-seller and great game that acquired a significant reputation for a horrible multiplayer experience because of cheaters. Many people I know either refused to play it online, or would only play over a LAN with trusted friends. Blizzard did their best to respond, patching it multiple times, but they were fighting an uphill battle.

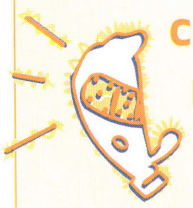
Cheating hit closer to home for me while I was working on the final stages of AGE OF EMPIRES II: THE AGE OF KINGS. Cheating online became a widespread problem with the original AGE OF EMPIRES. Tournaments had to be cancelled due to a lack of credibility, the number of online players fell, and the reputation of my company took a direct hit from frustrated users. Unable to spare the resources to fix the game properly until after AGE OF KINGS was done, we just had to endure our users turning their anger upon us — probably the most personally painful thing I've experienced as a developer.

What about your next game? This is a good time to introduce my first two rules about online cheating:

RULE #1: IF YOU BUILD IT, THEY WILL COME — TO HACK AND CHEAT.

RULE #2: HACKING ATTEMPTS INCREASE WITH THE SUCCESS OF YOUR GAME.

Need more reasons to take online cheating seriously? Go onto eBay and type in the name of your favorite massively multiplayer game. Now look at the real money changing hands for virtual characters and items. What if those items being sold were obtained via some sort of cheat or hack? Let's not overlook the growth of tournaments and contests for online games.



Cheaters Never Prosper

RULE #1: *If you build it, they will come — to hack and cheat.*

RULE #2: *Hacking attempts increase with the success of your game.*

RULE #3: *Cheaters actively try to keep developers from learning their cheats.*

RULE #4: *Your game, along with everything on the cheater's computer, is not secure. The files are not secure. Memory is not secure. Services and drivers are not secure.*

RULE #5: *Obscurity is not security.*

RULE #6: *Any communication over an open line is vulnerable to interception, analysis, and modification.*

RULE #7: *There is no such thing as a harmless cheat or exploit. Cheaters are incredibly inventive at figuring out how to get the most out of any loophole or exploit.*

RULE #8: *Trust in the server is everything in a client-server game.*

RULE #9: *Honest players would love for a game to tip them off to possible cheating. Cheaters want the opposite.*

Consider the public relations nightmare that would ensue if the winner of a cash prize in a tournament had cheated. Enough to give you a headache, eh?

Understanding the Hackers and Cheaters

The sad truth is that the Internet is full of people that love to ruin the online experiences of others. They get off on it. A great many cheaters use hacks, trainers, bots, and whatnot in order to win games. But while some openly try to wreak havoc, many really want to dominate and crush opponents, trying to make other players think they are gods at the game — not the cheaters they are. The only thing that seems to bother them is getting caught. Beyond that, no ethical dilemmas seem to concern them. The anonymity and artificiality of the Internet seems to encourage a moral vacuum where otherwise nice people often behave in the worst possible way. A big factor in this is a lack of consequences. If a player is caught, so what? Are they fined or punished? No. Are they rejected by the people they played against? Usually, but it's so easy to establish another identity and return to play that discovery and banishment are no barrier to those with ill intent.

Another interesting aspect of online cheating is the rise of clans and how cheats get propagated. If a member of a clan hacks a game or obtains a not-readily-available program for cheating, it will often be given to other members of the clan with the understanding that it's for clan use only and to be kept secret. The purpose being, of course, to raise the standing and prestige of the clan. If the cheater is not a clan member, odds are he will keep the secret to himself for a while and not advertise his advantage. The logic here is simple: If anyone goes public with a cheat, a) he will lose his advantage, b) he will probably be identified by his opponents as a cheater, and c) the developer can then patch the game, invalidating the cheat. As a result of this secretive behavior we get to rule number three.

> RULE #3: CHEATERS ACTIVELY TRY TO KEEP DEVELOPERS FROM LEARNING THEIR CHEATS.

Tools of the Hackers

So how do they discover the hacks and create the programs to cheat at your game? Consider rule number four:

> RULE #4: YOUR GAME, ALONG WITH EVERYTHING ON THE CHEATER'S COMPUTER, IS NOT SECURE. THE FILES ARE NOT SECURE. MEMORY IS NOT SECURE. SERVICES AND DRIVERS ARE NOT SECURE.

That's right, you gave them a copy of your game when they purchased it. The hackers have access to the same tools that you had while making the game. They have the compilers, dissemblers, debuggers, and utilities that you have, and a few that you don't. And they are smart people — they are probably more familiar with the Assembly output of an optimized C++ file than you are. The most popular tool among the hackers I surveyed was NuMega's excellent debugger, SoftIce — definitely not a tool for the wimpy. On another day, you just might be trying to hire these people. Many of them possess a true hacker ethic, doing it just to prove it can be done, but more do it specifically to cheat. Either way we get the same result: a compromised game and an advantage to the cheater.

Hacking games is nothing new, it's been going on as long there have been computer games. For single-player games, it has never been an issue, since no matter what a player does with a game, he's only doing it to himself (and therefore must be happy about it). What's new is bringing the results of the hacking to other players, who never wanted or asked for it.

I've lost count of the number of developers I've encountered who thought that because something they designed was complicated and nobody else had the documentation, it was secure from prying eyes and hands. This is not true, as I learned the hard way. If you are skeptical, I invite you to look at the custom graphics file format used in AGE OF EMPIRES. Last year, I received a demanding e-mail from a kid who wanted the file format for a utility he was writing. I told him to go away. Three



You can't hide from change. But you can prepare for it. Embrace it even. That's where we come in. With an eMarketplace that matches your unique skills with the most rewarding opportunities. With powerful tools that help you run your business better. And with a virtual network that connects you with the right kind of companies — those that can see as far ahead as you.

SkillsVillage.com™

A new force at work.

FIGURE 1. Cheating classifications.

- Reflex augmentation
- Authoritative clients
- Information exposure
- Compromised servers
- Bugs and design loopholes
- Environmental weaknesses

days later he sent me the file format documentation that he reverse-engineered, and asked if he missed anything. He hadn't. Thus, this is a perfect example of rule number five. Yes, I've borrowed it from cryptography, but it applies equally well here.

RULE #5: OBSCURITY IS NOT SECURITY.

Sometimes we do things, such as leaving debug information in the game's executable, that make the hacker's job easier. In the end, we cannot prevent most cheating. But we can make it tough. We don't want effective cheating to be a matter of just patching six bytes in a file. Ideally we want hacking a game to be so much work that it approaches the level of having to completely rewrite the game — something that goes outside the realm of any reasonableness on the hacker's part.

One of biggest things we often do that makes it easier for a hacker, and thus harder on us, is include Easter eggs and cheat codes in the single-player portion of our games. Considered to be practically a requirement, they expose extralegal capabilities of our game engines and make it much easier for the hackers to locate the data and code that controls that functionality.

Models of Multiplayer Communications

Most online games use one of two communication models: client-server and peer-to-peer. For our discussion, the deciding factor is where game event decisions are made. If only one player's (or a separate) computer makes game event decisions or has the game simulation data, it is client-server. If all players' computers make some or all of the game event decisions, or have the full game simulation, then it's peer-to-peer. Many of the cheating methods described here are applicable to both models. I've organized the various cheats, trainers, exploits, and hacks that I've learned about into the categories listed in Figure 1.

Oh Look, It's the Terminator...

The first type of cheat is reflex augmentation, which is when a computer program replaces human reaction to produce superior results. This type of cheating is really only applicable to games where reflexes and reaction times matter, and thus is most applicable to action games.

During my FPS obsession, I believe that I encountered a form of reflex augmentation known as an aiming proxy. An FPS aiming proxy works like this: The proxy program is run on a networked computer and the player configures it with the address of the server they are going to play on. They then run the FPS game on another machine and connect to the proxy machine, which in turn connects the game to the server, acting just like an Internet packet router.

The only hitch is that the proxy monitors and attempts to decode all of the packets it is routing. The program keeps track of the movements and locations of all the players the server is reporting to the game, building a simple model. When the proxy sees a Fire Weapon command packet issued by the cheating player, it checks the locations and directions of all the players it is currently tracking and picks a target from them. It then inserts a Move/Rotate command packet into the stream going to the server in front of (or into) the Fire Weapon command packet that points the player straight at the selected target. And there you have it: perfect aim without all the mouse twisting.

When aiming proxies for QUAKE first appeared a couple of years ago, their targeting wasn't too sophisticated and didn't take into account things such as the player's field-of-view (FOV) or lag. Giveaways, such as players shooting weapons out of their backs, tipped people off that something foul was afoot. One of the first countermeasures to be developed was a server add-on that statistically identified players whose aim was too good to be true, then kicked out and banned the perpetrators. This naturally proved controversial, since some people really are "railgun gods," and the issue of possibly falsely identifying a person as a cheater was raised (and has yet to go away). And of course, the aiming proxies evolved with time. Later versions were improved to consider only the player's current FOV and compensate for lag, and added just enough randomness in their aim to stay below a server's "too good to be legit" identification threshold.

This big vulnerability is summed up in rule number six. Since the proxy is not running on the same computer as the game client, definitive detection can be next to impossible. Making the development of the proxy extremely difficult then becomes a priority.

RULE #6: ANY COMMUNICATION OVER AN OPEN LINE IS VULNERABLE TO INTERCEPTION, ANALYSIS, AND MODIFICATION.

One way to inhibit this form of cheating is to encrypt the command packets so that the proxies can't decode them. But there are limits to the extent that encryption can be used on communications. Most FPS games can send and receive a couple of kilobytes of data or more per player per second, and have to allow for lost and out-of-order packets. The encryption therefore has to be fast enough not to impact frame rate, and a given packet's encryption can not be dependent on any other packet unless guaranteed delivery is used. And once the encryption is cracked, the game is vulnerable until the encryption is revised, which usually involves issuing a patch. Then the hacking starts over.

Another way to make life more difficult for the proxy creator is to make the command syntax dynamic. Using something as simple as a seed number that's given to the game when it connects and a custom random number function, the actual opcodes used in the communication packets can be changed from game to game, or even more often. The seed itself doesn't have to be transmitted; it could be derived from some

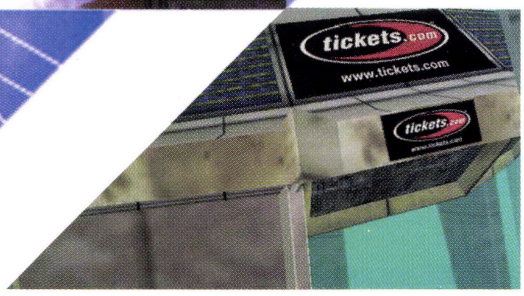
It's more than a game.
It's how you make your living.

Make both better with Adaboy.

Adaboy places ads seamlessly into the worlds you create without interrupting game play or stealing space from the screen.

Targeted Message Technology™ from Adaboy. It makes games better. It makes you money.





For more information, contact:
Adaboy, Inc. • 622 Second Avenue • Pittsburgh, PA 15219
gamers@adaboy.com Or visit www.adaboy.com

Adaboy™

All trademarks and registered trademarks are the property of their respective owners.

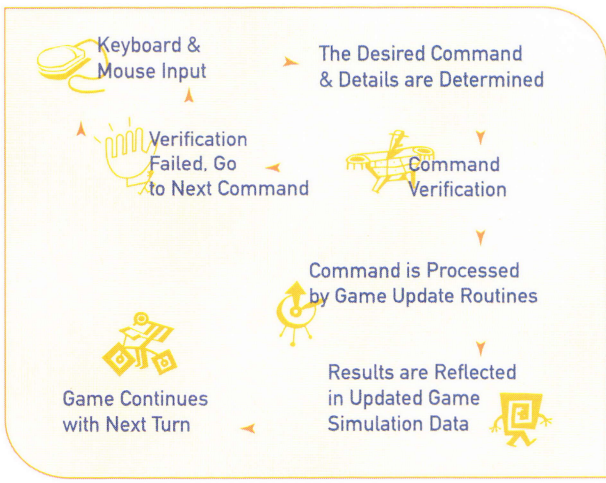


FIGURE 2. Single-player-game command processing steps.

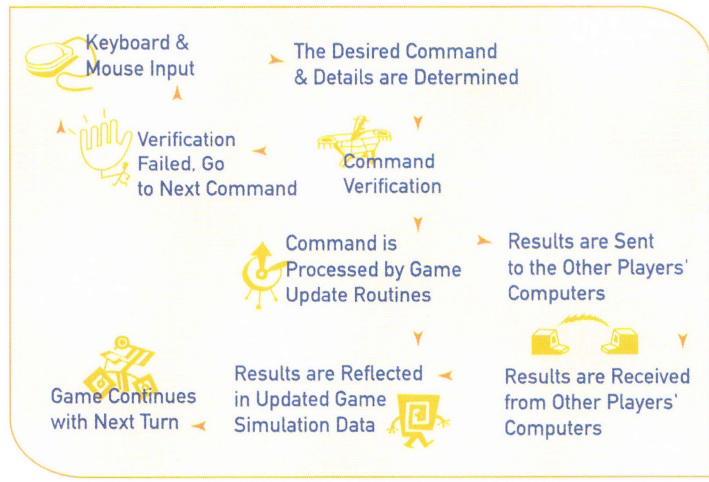


FIGURE 3. Single-player-game command processing extended to support multiplayer operation.

aspect of the current game itself. The idea here is that since a proxy sees all the communications, but *only* the communications, the random seed is derived from something not explicitly communicated. Foolproof? No. But it's far more difficult to hack, forcing the hackers to start from scratch.

If guaranteed delivery is used, another communications protection technique is to serialize each packet. Taking it a bit further, you could make a portion of the next serial number dependent on a checksum of the last packet. While there are speed issues with the delivery, it's an excellent way to make it difficult to insert or modify packets.

Though reflex augmentation seems to be exclusive to FPS games, the vulnerability extends to any game where quick reflexes can make a difference and game communications can be sniffed.

The Client Is Always Right

The next major class of cheats is exploiting authoritative clients. This is when one player's modified copy of an online game tells all the other players that a definitive game event has occurred. Examples of the communications would be "player 1 hit player 2 with the death-look spell for 200 points of damage," "player 2 has 10,000 hit points," and so on. The other players' games accept these as fact without challenging them and update their copy of the game simulation accordingly.

In this case, a hacked client can be created in many ways: The executables can be patched to behave differently, the game data files can be modified to change the game properties on the hacked client, or the network communication packets can be compromised. In any case, the result is the same — the game sends modified commands to the other players who blindly accept them. Games are especially vulnerable to this type of exploit when they are based on a single-player game engine that has been extended to support online multiplay in the most direct (read: quickest to develop) manner.

Fortunately there are several steps that a game developer can take to eliminate most problems with authoritative clients. A first step is to install a mechanism in the game that verifies that each player is using the same program and data files. This means going out and computing a CRC or similar identifier for all the data in question, not just relying on a value stored in the file or the file size. A nice side benefit is that this method also detects out-of-date files during the development process.

For peer-to-peer games, cheating can be made difficult by changing from a game engine that issues commands to one that issues command requests. It's a subtle distinction but one that requires engineering changes throughout the game. It also requires that each player's machine run a full copy of the game simulation, operating in lockstep with the other players.

Command processing in a single-player game typically works in the manner shown in Figure 2. The player issues some sort of command via the game's user interface. The game then performs a validation check on the command to see if the player has the resource, the move is legal, and so on. The game then performs the command and updates its internal game simulation. Figure 3 shows game engine command processing extended to support multiple players in the most direct way possible. The process stays the same except for the addition of a communications packet that's sent out to inform the other players of what has taken place. The receiving players integrate the data directly into their world simulation.

With the shift to command requests, the order of events changes a bit, which is shown in Figure 4. After determining that the command is a legal one, a command request describing the command is sent out to other players and is also placed into the player's own internal command queue, which contains command requests from other players as well as his own requests. Then the game engine pulls command requests from the queue and performs another validation check, rejecting the request if it fails. The fundamental difference is that every player has a chance to reject every action in the game based solely on the information on that player's machine. No other machine provides the information to make the determination on what is right and wrong. A hacked game



PlayStation is a registered trademark of Sony Computer Entertainment Inc.

PRODG

It's no understatement to say PlayStation® 2 will be a great adventure. And like all great adventures, you begin with a mixture of excitement and trepidation. Pro-DG, the complete development toolkit, has been created to ensure you get the most out of it. Pro-DG is fully customisable, works at your speed and expands your design capabilities. Register for your free evaluation copy now at **www.snsys.com/gameon**

MOVING THE GAME ON

cannot reach out and alter what's on an honest player's machine with this approach. Note that such an architecture works equally well for a single-player game.

Preventing a dishonest command from being accepted on an honest player's machine is only half the task. The game also has to be able to determine whether someone is playing the same game and if not, it must do something about it. For instance, when a received command request is rejected for reasons that should have prevented it from being issued in the first place (remember, the issuer is supposed to have checked it for validity before passing it to the other players), all other players should assume that a cheater is in their midst, and take some sort of action.

Often though, due to design issues (such as posting command requests to a future turn), it is not possible to thoroughly ensure that all command requests passed to other players won't be rejected if a player is being honest. A good way to deal with this is to add synchronization checking to the game. At various points during the game, each player's machine creates a status summary of the entire game simulation on that computer. The status, in the form of a series of flags, CRCs, and checksums, is then sent to all the other players for comparison. All the status summaries should be the same, provided the game program and data files are the same for each machine. If it turns out that one player has a different status from all the rest, the game can take action (like drop the player from the game). The idea is that a hacked game should cause that player's game simulation to produce different results.

Alternatively, you can make life even more difficult for the hacker by easing up on the received command request evaluations. By allowing command requests to bypass the verification check only on the machine that issued it, you're deliberately allowing the game to go out of synch if the initial verification check or data has been hacked. Combine this with a synchronization check that occurs somewhat infrequently and you've presented the hacker with something of a mystery — on his machine the cheat worked, but then a while later the other players booted him out of the game.

This status synchronization has a huge benefit for the development process as well. Getting a complicated game engine to produce the same game simulations results while having different player views, inputs, and settings is a very difficult task. It's difficult to keep the simulation-independent code from accidentally impacting the simulation. For example, a compare against the current player number variable in the simulation code, or randomly playing a background sound based on an object in the player's view using the same random function used by the simulation, could cause future executions to produce different results on different machines. Judicious use of status synchronization allows a developer to quickly narrow down the portion of the game that isn't executing the same for all players.

Client-server games unfortunately can't benefit as much from these techniques, as they lack full game information and by design must rely on the authority of the server. We will look at this more a bit later.

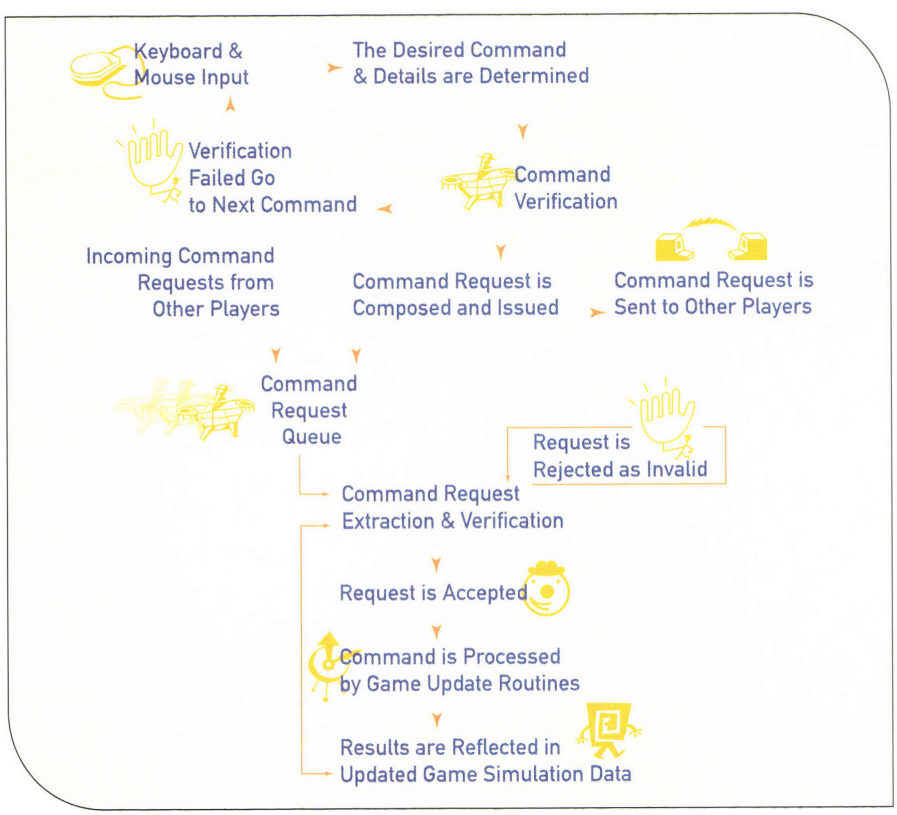


FIGURE 4. Command processing steps when using command requests.

The Game Emperor's New Clothes

The next major class of cheats is what I've dubbed "information exposure." The principle is simple: On a compromised client, the player is given access or visibility to hidden information. The fundamental difference between this and authoritative clients is that information exposure does not alter communications with the other players. Any commands sent by the cheater are normal game commands — the difference is that the cheater acts upon superior information.

The first-person-shooter cheats of modified maps and models arguably fall under this classification, as they let cheating players see things that they normally wouldn't be able to (in the case of modified maps), or see them more easily (in the case of a modified player model that glows in the dark). Any game whose game play relies on some information being hidden from a player has a lot to lose to these types of cheats.

The real-time strategy (RTS) genre suf-

fers severely from this. The most obvious being hacks that remove the "fog of war" and "unexplored map" areas from the display. With a fully visible map, the cheating player can watch what other players are planning and head them off at the pass, so to speak.

There are a couple of ways the hacker accomplishes this. The hacker may go after the variables that control the display characteristics of the map. With the help of a good debugger and single-player cheat codes to reveal the whole map, finding the locations in memory that control the map display is fairly simple. Then either the game .EXE file is modified to initialize those map control values differently, or a program is made that attaches to the game's memory space and modifies the variable values while the game is running. To combat this, the values of those variables should be regularly reported to other players in the form of a checksum or CRC code. Unfortunately, that only raises the stakes; the hackers then just attack the code that reads those control values (easy enough to find quickly), inverting or `NOP`'ing out the instructions that act upon them.

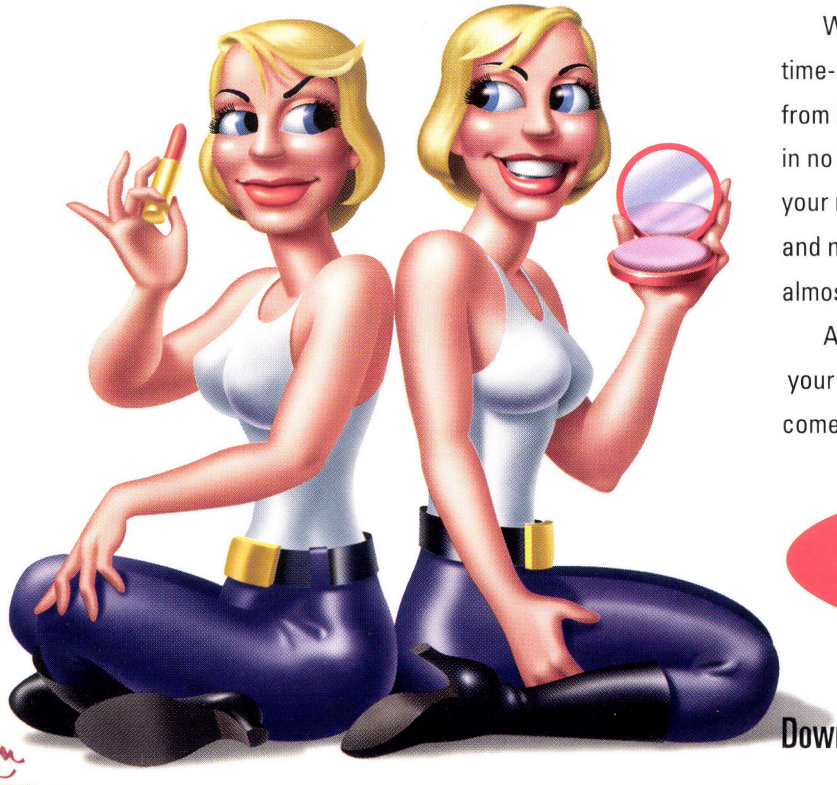
Additional techniques are needed to detect the hacked game view. There are a couple of ways to take advantage of the fact that the full game simulation is run on all clients. One way is to borrow a technique from the "authoritative client" section and check each command request for the side effects of a hacked map on one of the players. We specifically ask the game simulation, which is separate from the screen display, the question, "Can that player see the object he just clicked on?" In doing this we are assuming ahead of time that such hacks will be attempted, making sure we consider the side effects by which they might be detected. Once again, easing up on checks of the player's own machine is very useful. The next time the game performs a synchronization check, all the other players will agree that the cheating client is "out of synch" with the rest of the game and can deal with him accordingly.

Another technique that avoids looking at the display control variables is to compile abstract statistics on what gets drawn to the screen. The statistics are derived from the game simulation data and just filed away. This doesn't immediately prevent the hacker from cheating; instead, you send the statistics around as part of the status synchronization and see what the other players think of them.

In the RTS map-hack case, it is necessary for some change to be made to the game; either the code or some data is in a modified state while the game is running. And if something has been modified, you can attempt to detect that.

But information exposure cheats can be totally passive. Consider a scenario where a program gains access to the memory space of an RTS game that is running. It then reads key values for each player in the game out of memory and sends them to an adjacent networked computer. An industrious hacker once raised that sce-

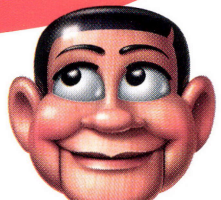
240 hours of modeling. 110 hours of texturing.
83,231 painstaking polygons. And they still won't talk...



Who ever said automated lip-synching had to be a time-consuming and expensive process. With Ventriloquist from LIPSinc your creations can be talking a mile a minute in no time at all. And talk about effective! For the first time your models will look like they are actually saying the words and not just moving their mouths—an accomplishment almost unheard of before Ventriloquist came along.

At LIPSinc we believe that if you put a little money where your mouth is, it will pay off in your final product. When it comes to lip-synching, Ventriloquist says it all.

VeNTriLOQUiST®
LIPSINC.



Download Now! www.LIPSinc.com

LIPSINC.

nario with me regarding one of the AGE OF EMPIRES games, saying he had figured out how to read out of memory the resource amounts for every player. At first we thought that this wasn't very serious. He then explained that if he polled the values a couple hundred times a second, he could identify nearly every discrete transaction. A simple Visual Basic program could then display a log window for each player, with messages for events such as the training of various units (to the extent they could be distinguished from others on the basis of cost), and messages for events such as building construction, tribute, and advancement to the next age. Basically, this cheating method was the next best thing to looking over the shoulders of his opponents.

RULE #7: THERE IS NO SUCH THING AS A HARMLESS CHEAT OR EXPLOIT. CHEATERS ARE INCREDIBLY INVENTIVE AT FIGURING OUT HOW TO GET THE MOST OUT OF ANY LOOPHOLE OR EXPLOIT.

Intrigued, I asked him how he could be sure he had found the correct memory locations each time, as they changed each game since they were stored in dynamically allocated classes. His answer was most interesting. He first scanned the memory space of a paused game looking for known values for things such as population, wood, gold, and other very significant game values that he knew about and believed were unique. He had a simple custom program that looked for the values in basic formats such as long ints and floats. After his program identified all the possible addresses with those values, he ran the game a bit more until the values had changed. He then reran the program, checking the prior list of locations for the new values, reducing the list of possible addresses until he was sure he had found the correct locations. He then put a read-access breakpoint on the value and looked at how it was accessed from various points in the code. At one of the breakpoints, the C++ code for accessing the wood amount looked something like this:

```
GAME_MASTER -> GAME_WORLD->PLAYER[n].
ResourceAmount[Wood_Index];
```

LISTING 1. Hiding the variables that tip off hackers to possible cheats.

```
void GameResource::SetResource(int Resource_Num, int Resource_Amount)
{
    GameResourceAmount[ResourceNum] = Resource_Amount ^ EncryptValue[ResourceNum];
}
int GameResouce::GetResource(int Resource_Num)
{
    return( GameResourceAmount[ResourceNum] ^ EncryptValue[ResourceNum] );
}
//and more specific functions...
void GameResource::SetWood(int Wood_Amount)
{
    GameResourceAmount[RESOURCE_WOOD] = Wood_Amount ^ EncryptValue[RESOURCE_WOOD];
}
int GameResource::GetWood(void)
{
    return( GameResourceAmount[RESOURCE_WOOD] ^ EncryptValue[RESOURCE_WOOD] );
}
```

This is a pointer to a pointer to an object containing an array of integers, one of which contains the value of the player's current stockpile of wood, and all the objects are dynamically allocated. The hacker's point was that if you trace back through all the dynamic pointers, you eventually find a static variable or base pointer. The different spots where his breakpoints were triggered were from member functions at different levels in the class hierarchy, and even from outside the class hierarchy containing the data. And it was finding an instance of that latter access condition that was the jackpot. There it was in his debugger's disassembly window: a base address and the Assembly code to traverse through the classes and handle player and resource index numbers.

Considering all this, I found a couple of strategies that can greatly reduce the likelihood of this sort of passive attack. Again, these tips cannot guarantee 100 percent security, but they make the hacker's job much harder.

The first strategy is to encrypt very significant values in memory at all times. Upon consideration, most game variables are not significant enough to warrant such protection — the hit points of a particular object don't tell anyone much, while a drop of 1,000 food and 800 gold from a player's resources does indicate that the player is advancing to the Imperial Age, which is an event of large strategic importance in our game. Simple encryption is relatively easy when all access to the variables goes through assessor functions. A communicative function such as `XOR` is your friend here, as it alters values upon storing, restores them upon reading, and is extremely fast. The whole point is to make it very hard for the hacker to find the variables he is searching for in the first place. Values the hacker would know to look for are not left around so that a simple scan can find them. In C++, our encrypted assessor functions for game resources look something like what's shown in Listing 1.

The second strategy for slowing down passive attacks is to never access very significant values from outside the class hierarchy. Assuming the values are located while using the debugger, try not to access them in a way that starts with a reliably fixed memory address. Combining this with small, randomly sized spacing buffer allocations during the main game setup ensures that the memory addresses for vital information will never be the same from one game to the next. A piece of C++ code you *won't* see in our next RTS game would be the following:

```
GLOBAL_GAME_POINTER -> PLAYER_DATA[n] ->
RESOURCE_TABLE[gold] = SOME_UNIQUE_START_VALUE;
```

Information access isn't limited to games as complex as RTS games, it can extend to something as simple as a card game. Consider an online card game such as poker. All it would take to ruin the game is for a player to see the values of the face-down cards in another player's hand. If the information is on the machine, hackers can go digging for it. This goes back to rule number four.

Who Do You Trust Baby?

In client-server games, because so much is controlled by the server, the game is only as good as the trust placed in the server and those who run it.

RULE #8: TRUST IN THE SERVER IS EVERYTHING IN A CLIENT-SERVER GAME.

An issue here is brought up because some client-server games can be customized by the user running the server. Access and configurability are great for many games, as they allow the player community to extend and evolve a game. But some individuals will test the server to see what can be exploited in the name of cheating. This in itself is not the problem — rather it's when honest but unaware players find their way to the server and don't know that they are not on a level playing field.

You really need to consider your audience here. A successful game will sell hundreds of thousands of copies, if not millions. You as a developer will be most in tune with the hard-core players — those that know the game inside and out. But it's easy to forget about the more casual players, who probably will be the majority of purchasers of your game once you pass a certain level of success. These are the people who don't know to check the status of the `Cheats_Allowed` flag before joining a server, or that game rule changes are transparently downloaded when they connect. All they probably know is the default game configuration, and when they see their ReallyBFG27K gun doing only 0.5 points of damage, they're going to cry foul. It doesn't matter that it was technically legal for the server operator to make the change, you still wind up with a user that is soured on the game and not likely to recommend it to his buddies anymore.

Naturally, people get a whole lot more unhappy with a game when they encounter modifications with malicious intent. What if a clan decided to add a tiny server mod to their FPS server that looked something like this snippet of C code:

```
If Player.Name->Contains("OUR_CLAN")
Taken_Damage = Taken_Damage * 0.80;
```

Or what if the remote console was hacked to allow normal cheats to be toggled? Dishonest players in with the server could make a key-bind that resembled this:

```
Access_password on; set Cheats_Allowed true;
Give Big_Ass_Weapon; Give Big_Ass_Ammo; Set
Cheats_Allowed false; Access_Password off;
```

The important point here is that with user-run servers and powerfully configurable game engines, these kinds of shenanigans will happen. While we as developers can't protect our more casual users from joining any game server they wish, we can do a better job of letting them know when they are encountering something that could be different from what they expect. QUAKE 3: ARENA set a great example when it introduced the concept of a "pure" server. It's a simple idea that casual users can quickly grasp and set their gaming expectations by.

But why stop there? If we download data that includes a new set of weapon properties, why not put a message on the screen saying, "Weapon properties modified"? If single-player cheat commands are issued in the middle of a game, maybe we should send a message to every client notifying them of that fact, so even players who aren't near the issuer can be made aware. Empower players to easily determine whether the games are fair or not.

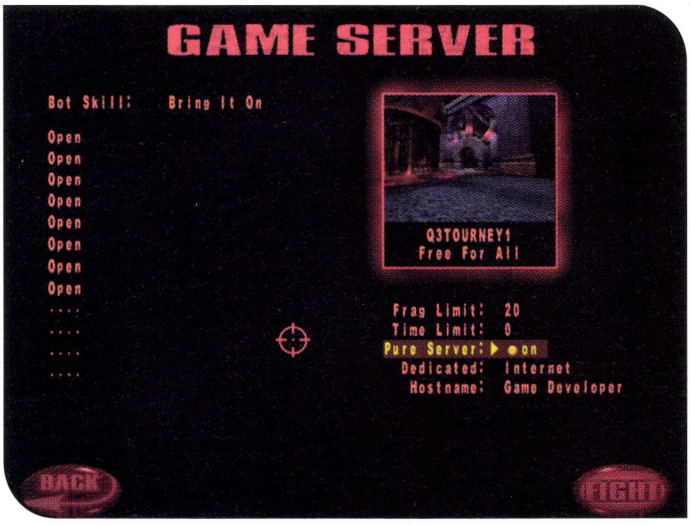
RULE #9: HONEST PLAYERS WOULD LOVE FOR A GAME TO TIP THEM OFF TO POSSIBLE CHEATING. CHEATERS WANT THE OPPOSITE.

Bugs & Design Issues

Technically, this category of cheats is one that we bring upon ourselves: bugs in our games can be discovered by users and used to disrupt game play. Most bugs don't enable cheating, but a few do.

A good example is the farm-stopping bug in the unpatched version of AGE OF EMPIRES. When a user had both a villager and a farm selected, he could issue the Stop command. Because the command was valid for a villager, it was allowed to go through, but listed both objects as the target of the command. The villager would stop working as expected and reset its state. The farm would also reset itself, something it never did normally, and replenish its food supply. Once this was discovered by players, it drastically changed the game for them, giving them a huge advantage over those who didn't know about it.

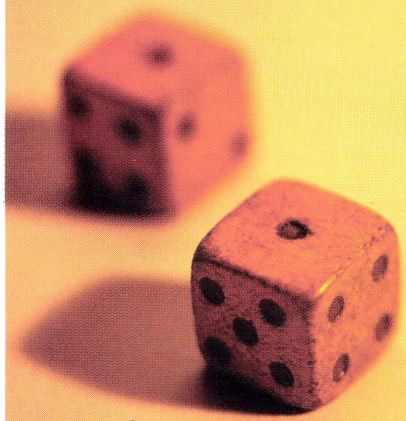
I encountered another bug when playing HALF-LIFE. I would get into a firefight with



QUAKE 3: ARENA introduced the concept of a "pure" server so players know what to expect when joining a multiplayer game.

We'd like you to take on 30 million Americans.

Believe it or not, that's how many play dice every day. Which is why we've created a set of real dice that work seamlessly with both computer and video games.



CIBRO Technologies Ltd.

Are you ready to roll?

www.cibrodice.com
email: info@cibrodice.com

another player, both of us using the same weapon, but when it came time to reload our weapons, my opponent was able to reload much more quickly than I could. Sure enough, when the next patch came out, I saw in the release notes that a bug allowing fast reloads was fixed. There's really not much we can do about these types of bugs, other than fix them with a patch.

Environmental Weaknesses

My last category of cheats is something of a catchall for exploitable problems a game may have on particular hardware or operating conditions. A good example is the "construction-cancelled" bug that was found amazingly in both AGE OF EMPIRES and STARCRAFT at about the same time. The element needed to make it work was extreme lag in network communications, to the point of a momentary disconnection. When this happened, the game engines stopped advancing to the next game turn while they waited for communications to resume. During this time, the user interface still functioned, so the player didn't think the game had locked up. While the game was in this state, a player could issue a command to cancel construction of a building, returning its resources to the player's inventory — only the player would issue the command over and over as many times as possible. Normally, a player could only issue one Cancel command per turn, but because the game simulation was in a holding state, multiple command requests went into the queue. Because of some necessities of RTS engine design, when an object is destroyed during a turn by something such as a Cancel command, the destruction is postponed until after all the commands for that turn have been processed. The result was the command executed multiple times during one game update.

Once discovered, this had a horrible impact on online games. People deliberately caused massive lags to take advantage of the cheat. To fix it in AGE OF EMPIRES, we had to update the validation checks to see if a similar request was already pending on the current turn and reject duplicates.

Another bug of this type involved the game FIRESTORM and its interaction with the Windows clipboard. It seems a clever user found out that if he pasted text from the clipboard into his chats and that text contained a certain character not normally used, the game would crash when it attempted to print it to the screen — on all player's machines. He then treated this knowledge as a personal nuclear bomb that he could spring on people when he found himself losing.

Yet another example taken from AGE OF EMPIRES is what happens when a player's network connection is overloaded or ping-flooded by another player. When such an attack renders a game unable to communicate with its peers, the other players decide that something is wrong with that player and drop him from the game — a totally necessary capability, but one that can be exploited in a modern twist on scattering all the pieces on a game board when you are losing. This was one the major reasons we added Multiplayer Save and Restore capabilities to AGE OF EMPIRES II.

Some Final Thoughts

I hope these examples got you thinking about some of the problems and issues at stake when developers address the problem of online cheating. We certainly have a lot more ground to cover, from massively multiplayer games, open source, and consoles, to enabling the online communities to better police the situation. But we're out of space and time for now.

ACKNOWLEDGEMENTS

Special thanks to the following people who provided valuable information and insight for this article:

- **Greg Costikyan** *author of "The Future of Online Gaming" (www.costik.com)*
- **Harvey Smith** *Ion Storm Austin*
- **Mark Terrano** *Ensemble Studios*
- **Gordon Walton** *Origin Systems*

ON THE FRONT LINE OF GAME INNOVATION

Game DEVELOPER

new!

Take the last year of Game Developer magazine for a spin!

May 1999 - April 2000



Get 12 issues of hard-to-find technical information, all on 1 CD-ROM.

It's fully searchable!

You depend on *Game Developer* magazine to inspire and empower your team, but sometimes finding a referenced technique from a particular article can be time consuming. Why spend hours flipping through pages when you can search through the entire year's worth of issues in seconds?



With the GAME DEVELOPER BACK ISSUES CD-ROM, Volume II, you have all the gems the magazine has published over the past year right at your fingertips.

This disc contains every issue of *Game Developer* magazine from May '99 to April '00 in Adobe Acrobat format. You'll find quality, cut-to-the-chase articles written by and for professional game developers, information that you just can't get anywhere else. Imagine the lessons you'll take away from recent postmortem articles, including Dreamworks' Trespasser, Looking Glass's Thief: The Dark Project, and Westwood's Command & Conquer: Tiberian Sun. Take advantage of years of shared wisdom from experts like Chris Hecker, Doug Church, Saxs Persson and many more.

FEATURES

- **FULL ACCESS TO ARTICLES RIGHT ON YOUR DESKTOP.** You get the full text of feature articles, columns, and product reviews from the May 1999-April 2000 issues.
- **VISUAL CLARITY.** Articles, artwork, and diagrams appear exactly as they were printed in the magazine, thanks to Adobe Acrobat.
- **EASY TO USE ACROSS ALL PLATFORMS.** It supports major platforms including Windows, Macintosh and Unix.
- **INCLUDES CODE AND APPLICATIONS** that accompany articles. It's all at your fingertips.

ONLY $49!

(Plus $5 S&H in the US. Add sales tax where applicable. Call 800.444.4881 for site licensing options)

SPECIAL OFFER — JUST $120 FOR BOTH CD-ROMS

Save more than 20% when you purchase both volumes 1 and 2 of the GAME DEVELOPER BACK ISSUES CD-ROM. Get 52 issues of the magazine (from the premiere issue in 1994 through the April 2000 issue) Just $120 for both CD-ROMs! (Plus $6 S&H in the US. Add sales tax where applicable.)

Order this unique reference today!

CALL 800.444.4881, in the US
CALL 785.841.1631, outside the US
FAX 785.841.2624
EMAIL orders@mfi.com

or send orders to: GAME DEVELOPER BACK ISSUES CD-ROM, Volume II
1601 West 23rd Street, Suite 200 ▸ Lawrence, KS 66046-2703

Sponsored by:

Maya™
Alias|wavefront

CMP
Game MEDIA GROUP

Profiling, Data Analysis, Scalability, and Magic Numbers

Meeting the Minimum Requirements for AGE OF EMPIRES II: THE AGE OF KINGS

AGE OF EMPIRES II: THE AGE OF KINGS (AoK), a tile-based, 2D isometric, real-time strategy game, was built on the code base used in the original AGE OF EMPIRES (AoE) and extended in its RISE OF ROME expansion pack. In these games, players guide one of many civilizations from the humble beginning of a few villagers to an empire of tens or hundreds of military and non-military units, while competing against other human or computer-controlled opponents in single or multiplayer modes.

This is the first of a two-part article that describes the tips, tricks, tools, and pitfalls that went into raising the performance profile of AGE OF EMPIRES II: THE AGE OF KINGS. All of the techniques and tools used to measure and improve AoK are fully capable of improving the performance of other games.



AoK's 2D graphics pipeline added new features to AoE's original system, implemented with a combination of C/C++ and hand-coded Assembly.

Beginning the Diagnosis

In some ways, the AoK development team was fortunate because we had the benefit of an existing code base to work with. Many performance improvements went into AoE, including extensive optimization of its graphics drawing core, and this work gave us a good starting point for AoK.

Still, a significant amount of new functionality was added over the course of the sequel's two-year development cycle. This new functionality, as well as new requirements placed on existing functionality, meant that there was a large amount of new work to do in order to meet the minimum system requirements for shipping AoK. As such, a dedicated performance improvement phase began in April 1999 to ready AoK for its September 1999 release. The purpose of this phase was to identify and resolve the game's remaining outstanding performance issues, and to determine whether AoK would perform well on the intended minimum system configuration.

Our team had some ideas as to which parts of the code were taking a long time to execute, and we used Intel's VTune, NuMega's TrueTime, and our own profiling code to verify these hunches and see exactly where time was being spent during program execution. Often these performance results alone were enough to determine solutions but sometimes it wasn't clear why the AoK code was underperforming, and in these cases we analyzed the data and data flow to determine the nature of the problem.

Once a performance problem is identified, several options are available to fix it. The most straightforward and recognized solution is direct code optimization by optimizing the existing C code, translating it to hand-coded x86 Assembly, rearranging data layouts, and/or implementing an alternative algorithm.

Sometimes we found that an algorithm, though optimal for the situation, was executing too often. In one case, unit pathing had been highly optimized, but it was being called too often by other subsystems. In these cases, we fixed the problem by capping the number of times the code could be called by other systems or by capping the amount of time the code could spend executing. Alternately, we might change the algorithm so its processing could occur over multiple game updates instead of all at once.

AUTHOR'S BIO | *Herb Marselas currently works at Ensemble Studios. He helped out on AGE OF EMPIRES II: THE AGE OF KINGS. Shhhh! Please don't tell anyone he's working on a secret 3D-engine project called [deleted]. Previously, he worked at the Intel Platform Architecture Lab where he created the IPEAK Graphics Performance Toolkit. You can reach him at hmarselas@ensemblestudios.com.*

We also found that some functionality, no matter how much we optimized it, still executed too slowly. For example, supporting eight players in a game required too much processor time on the minimum system, so we specified that the minimum system could support only four players. We presented scalability features such as this as facets of game play or as options that players could adjust to their liking. These scalable features ultimately allowed AoK to run well on its stated minimum system, providing incentives or rewards to users who have better computers.

And then there were AoK's approximately 30 single-player scenarios. We evaluated the performance of these scenarios slightly differently from other game functionality. Instead of trying to optimize offending code, we first examined the scenario for performance problems that had been inadvertently introduced by the scenario designers in their construction of the scenario and it's elements. In many cases, performance improved significantly with slight changes to the scenario, for example reducing the number of player units, shrinking the game map, or making sections of the maps inaccessible to players.

FIGURE 1. AoK minimum PC and play-test PC system configurations.

MINIMUM SYSTEM SPEC TEST PC
- 133MHz Pentium processor*
- S3 Virge 2MB graphics card
- 32MB RAM
- Windows 98

ENSEMBLE PLAY-TEST PC
- 450MHz Pentium II processor
- Nvidia TNT 8MB graphics card
- 128MB RAM
- Windows 98

*later upgraded to 166MHz

FIGURE 2. AoK minimum system game play specifications.

- 4 players: any combination of human and computer players
- 4 players map size
- 75 unit population cap
- 800×600 resolution
- Low-detail terrain graphics quality*

*added as part of scalability effort

Above all, we made sure that we did not change the designer's vision of the scenario as we optimized it.

Shopping for Old Hardware

One of the goals of AoK was to keep the system requirements as low as possible. This was necessary in order to reach the broadest audience possible and to stay on the same incremental processor performance ramp set by the original AGE OF EMPIRES and its RISE OF ROME expansion pack. Our overriding concern was to meet these minimum system requirements yet still provide an enjoyable game experience.

The original AGE OF EMPIRES was released in September 1997 and required a 90MHz Pentium processor with 16MB RAM and a 2D graphics card capable of handling 8-bit palletized color. The RISE OF ROME expansion pack shipped a year later and raised the minimum system processor to a 120MHz Pentium. Based on this information, the AoK minimum processor was pegged as a 133MHz Pentium with 32MB of physical RAM (Figure 1). The additional RAM was required due mainly to the increased size and number of graphics and sound files used by AoK. There was also a greater amount of game data and an executable that grew from approximately 1.5MB for AoE to approximately 2.4MB for AoK.

To make sure AoK worked on the minimum system, we had to shop for old hardware. We purchased systems matching the minimum system specification from a local system reseller — we no longer used systems that slow. When the "new" computers arrived, we decided not to wipe the hard drives, nor did we reinstall software and hardware with the latest driver versions. We did this because we expected that most AoK users wouldn't optimize their computer's configuration or settings, either. Optimizing these systems would have undoubtedly improved our performance numbers, but it would not have translated into true performance gains on other minimally-configured computers. On the other hand, for normal in-house play-testing we used computers that were significantly more powerful than the minimum system configuration, which made up for performance issues caused by unoptimized code and enabled logging functions during play-testing (Figure 1).

A precedent set by the original AGE OF EMPIRES was the use of options and settings playable on the minimum system (Figure 2). A list of the specific options supported by the minimum system was needed due to the large number of them available in AoK (Figure 3). These were also the default options for the single-player and multiplayer games, and were used to guide the creation of approximately 30 single-player scenarios.

One of the first tasks of this dedicated performance phase was to determine the largest performance problems, the improvements that we could hope to make, and the likelihood that AoK would meet the minimum system specification in terms of processor and physical memory. This initial profiling process led us to increase the minimum required processor speed from 133 to 166MHz. We also felt that meeting the 32MB memory size could difficult but we were fairly certain that the memory footprint could be reduced enough to meet that goal.

Grist for Profiling

No matter how good or bad a program looks when viewed through the lens of profiling statistics, the only true test of satisfactory performance is how players feel about their game experience. To help correlate player responses with game performance in AoK, we used several on-screen counters that displayed the average and peak performance. Of these counters, the ones that calculated the average frame rate and lowest frame rate over the last several hundred frames were used most to determine performance problems. Additional statistics included average and peak game simulation time (in milliseconds) over the last several hundred game updates.

Identifying symptoms of play-testing performance problems and making saved games of these problem situations was very useful. We replayed saved games in the profiler and routines that took too long could be identified quickly. Unfortunately, some problems were difficult to track down, such as memory leaks and other programs running on the play-tester's computer.

We also created scenarios that stressed specific situations. For instance, we stressed the terrain engine's hill-drawing by using a special scenario consisting of a large game map covered mostly with hills. Other special scenarios were created that included many buildings, walls, or attempts to path units long distances between difficult obstacles. These scenarios were easy to build and it was obvious the first time the scenario was run whether a given issue needed to be targeted for optimization.

The final set of data to came in the form of recorded AoK games. AoK has a feature that allows human or computer player commands to be recorded to a file. This data can then be played back later as if the original player were issuing the commands. These recorded games helped diagnose pathfinding problems when it was unclear how a unit had arrived at a particular destination.

Since AoK was able to load scenarios, saved games, and recorded games from the command line, the game could be run automatically by a profiler. This simplified the profiling process by allowing the profiler to run AoK and have it jump directly into the problem. This command-line process bypassed the startup and pregame option screens. (Some profilers slowed the game down so much that manually loading a saved game from the profiler would have been impossible.) And since performance profiling and logging significantly slowed game play, analyzing recorded games was a much better solution from the tester's perspective. Multiplayer games could be recorded and then played back command-for-command under the profiler overnight to investigate performance issues.

Issues from the Original Age

Some performance issues from AoE needed to be resolved while we were working on AoK, the biggest of which was AoE's 2D graphics pipeline. The graphics for AoK are created through a combination of software rendering and hardware composition. This pipeline had been highly optimized for AoE by hand-coding most of the system in Assembly, so there was not much additional need to optimize it for AoK.

But there were new features to integrate into the 2D pipeline. For one thing, AoK had more detailed terrain. Also, units that were visually obscured behind buildings and other obstructions appeared as outlines so players could see where they were. Both of these systems were implemented as a mixture of C/C++ and hand-coded Assembly during implementation.

The biggest challenge in keeping the performance up for the graphics system was making sure that the sprites used for graphics in the game were properly tagged as belonging in system memory or video memory. If a sprite was in the wrong memory type a significant performance hit or even an error could occur, but it was usually hard to identify these graphics memory location problems. They were usually marked by a drawing problem on-screen, such as a shadow drawing on top of a unit instead of under it.

Sprites used by the software rendering engine needed to be in system memory so that they could be read and processed. If they resided in video memory instead, the limited throughput from video memory caused a significant performance hit to the game. Conversely, sprites bltted by the hardware that accidentally ended up in system memory would render slowly and could fail to render at all if the hardware bltter didn't support blts from system memory.

Pathfinding problems from AoE also had to be fixed. In AoE, there was a single unit-pathing system, which was known as "tile pathing" because it broke the game map down into individual tiles and classified them as "passable" or "nonpassable." This tile-pathing system was fairly good at moving units short distances, but it often took too long to find paths (if it could find one at all), so we created two additional pathing systems for AoK.

The first of these two systems, "MIP-map pathing," quickly approximated distant paths across the map. The basis for MIP-map pathing was the construction of compact bit vectors that described the passability of each tile on the map. This system allowed the game to determine quickly whether it was even possible for a unit to get from its current location to the general target area. The only way to determine whether the general area could be reached was through the resolution of the bit vectors.

Once a unit was within a short distance of its target area, another new pathing system called "low-level pathing" was used. Low-level pathing allowed very accurate pathing over short distances. When low-level pathing failed, the pathing system fell back and used the original tile pathing from AoE.

Changing the pathing system from a single, general-purpose system to three special-purpose systems improved the performance of AoK and also significantly improved game play since it virtually eliminated the

FIGURE 3. Game play and feature scalability.

NUMBER OF PLAYERS:	2 to 8, in any combination of human or computer
SIZE OF MAP:	2 to 8 player sizes and "giant" size
TYPE OF MAP:	• All land (Arabia) • Mostly water (islands) • Nine others in between (Coastal, Baltic, and so on)
UNIT POPULATION CAP:	25 to 200 units per player
CIVILIZATION SETS:	Western European, Eastern European, Middle Eastern, Asian
RESOLUTION:	• 800×600 • 1024×768 • 1280×1024
THREE TERRAIN DETAIL MODES:	• High detail — multi-pass, anisotropic filtering, RGB color calculation • Medium detail — multi-pass, fast lower-quality filtering, RGB color calculation • Low detail — single pass, 8-bit color lookup

- *new board of directors*
- *new management*
- *new resources*
- *new SIGS*
- *new local chapters*
- *new committees*

[new]

Join now to make a difference

COMPUTER GAME DEVELOPERS ASSOCIATION™

For further information, contact:
Liz Wakefield
Membership Coordinator
415.905.2474
info@cgda.org

www.cgda.org

problem of stuck and stopped units caused by pathing failures.

While we were able to improve the pathing system for AoK, enhancing the unit-class hierarchy system was a much more onerous task. The unit-class hierarchy system from AoE couldn't be changed easily since so many game systems and so much functionality relied on the old implementation. At its heart, the game's unit-class system is a hierarchy of derived classes and each derived class is more specialized than its parent. The special functions of each derived class are supported by virtual functions exposed by the classes in the hierarchy. A simplified version of the hierarchy is shown in Figure 4.

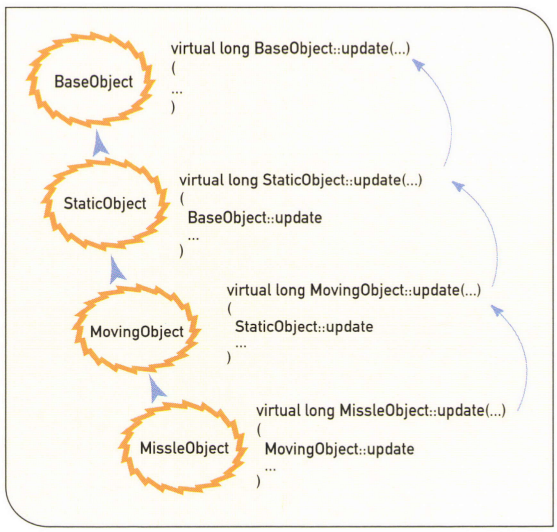


FIGURE 4. AoK unit class hierarchy.

From a programming standpoint, calling a virtual function consumes no more overhead than a regular class function. If each class could implement only its own version of the virtual functions, then this hierarchy wouldn't cause any function overhead problems. However, since each level of the hierarchy implements its own special code, it must also call its parent's version of the derived function to perform its work. In a hierarchy four classes deep, that means calling three additional functions. This may not sound like much, but it can add up when code is executed hundreds of thousands or millions of times.

Some performance improvement could have be gained by circumventing the hierarchy using "special case" units. For example, walls are a type of building unit that do not attack other units and only need to scan their vicinity for enemy units every few game updates unless they are under attack. To handle this special case, we could specifically check whether the current unit being processed is a wall, and if so, skip the code that is only executed for other buildings. Unfortunately, coding in too many special cases can also lead to performance losses, because you end up checking to see whether a unit is one of your many special cases. In the end, we left unit-class hierarchy in place, and made specific changes to shortcut to functionality that didn't apply to specific units.

Commercial Profiling Tools: The Good, the Bad, and the Ugly

Performance analysis extends beyond evaluating the execution speed of program functions and subsystems. It also includes measuring memory usage and evaluating the way the program interacts with other programs and the operating system. In order to determine where the performance problems were in AoK, four separate tools were used: Intel's VTune, NuMega's TrueTime, the Windows NT performance counters, and our own profiling and memory instrumentation code.

Although we used Microsoft Visual C++, we did not use the bundled Microsoft Profiler. There were two reasons for this: we found it difficult to get the Microsoft product to work correctly (or at all) and the data format from their profiler was either inadequate or needed post-processing in a spreadsheet to be minimally useful. Using VTune, TrueTime, and the NT performance counters we were able to collect, analyze, and present data in a reasonable fashion.

VTune is a sampling profiler, which means it has a component that wakes up every few milliseconds (or whatever amount of time you specify) and looks at what processes are executing on the CPU(s). When you decide enough time has elapsed, you can stop VTune and look at the statistics it produces for each process executed during that time. If you've compiled your program with debug information, VTune can display which lines of code were called and what percentage of the elapsed time was consumed by the executing code.

VTune is great because you don't need to compile a special version of your program, it doesn't slow your program down while it runs, and it lets you see the amount of time the CPU spent executing processes besides your own. The only major drawback is that you can end up with spurious data due to this sampling. This can be caused by other processes that are running in the system, or by running VTune for too long a period. To improve VTune's accuracy on your own program, it comes with an API to turn VTune on and off programmatically. This is a very useful feature, especially when drilling down into the performance of specific subsystems and smaller sections of code.

We found that VTune's call-graph functionality couldn't be used with a program that linked either explicitly or implicitly with DirectDraw. Also, some applications (including AoK) were too large in terms of code and debug information in order for VTune to resolve its data back correctly to a line of code. It seems that some of these problems have been fixed in VTune 4.5, however.

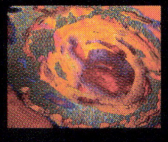
Another commercial product that we used was NuMega's TrueTime, which is an instrumenting profiler. To use this product, you have to make a special TrueTime compilation of your program that inserts timing code into each module. This can sometimes be a slow build process, but it's worth it. As the TrueTime build of your program runs, TrueTime logs which functions are entered, when they are entered, and when they are exited. This process can be significantly slower than VTune's effectively real-time performance but it's a useful second opinion nonetheless. The only big drawback (and it can be very severe) is that TrueTime can slow down your program so much that it's impossible to use it for profiling network code. This problem can also skew profiling statistics for time-based game actions such as AI or updates

Breakthroughs. Interactions. Insights. Visions. Applications. Adventures.









NEW ORLEANS

2000

SIGGRAPH

The world's premiere
conference and exhibition
on computer graphics
and interactive techniques.

Conference 23-28 July 2000
Exhibition 25-27 July 2000
www.siggraph.org/s2000/

Images (top to bottom): Civilization of Fruit: Martyred Apple © 1998 Patricia Swain; Still Life © 1998 Scott Curtin, Ringling School of Art and Design; Rolie Polie Olie © 1999 Nelvana; The Space Field Series: Comet Nebula © 1997 Victor Raphael

Sponsored by ACM SIGGRAPH

FIGURE 5. Performance analysis.

TEST PC 1	TEST CASE 1
166MHz Pentium	60 seconds of game play
32MB RAM	Eight-player game
S3 Virge GX	Giant map, largest map available
Windows 98	One civ from each of the for civ art sets
TEST PC 2	**TEST CASE 2**
Dual 450MHz Pentium III	60 seconds of game play
128MB RAM	Four-player game
Nvidia TNT2 Ultra	Four-player map size
Windows 2000	All civs share same civ art set

that are scheduled to occur at a certain interval of time.

This performance hit from TrueTime also made it impractical to use it to analyze the performance of the graphics subsystem. When system performance relies on two independent processors (such as the main CPU and the graphics card), efficient cooperation between both processors is critical so that they run concurrently and perform operations in parallel. When TrueTime slowed the CPU (and consequently the AoK rendering load which the CPU governed), it made the graphics card appear to give better performance than it actually did.

There were four drawbacks to both programs. First, neither program can be run in batch mode, so the programmer has to baby-sit the programs while they run through each performance test case. Even though we worked on performance test cases one at a time, it would have been convenient to run each program in batch mode overnight to gather results from other test cases. VTune has since added a batch interface in version 4.5 but support is still lacking in TrueTime.

Second, performance numbers gathered during the execution of a program need to be taken with a grain of salt. Due to the multi-threaded nature of the Windows operating system, other programs (including the performance tool itself) are effectively running at the same time, and that can skew performance. Fortunately, multiple performance runs with the same tool or with different tools can help identify specific problem areas by correlating all of the results, and analyzing performance over smaller sections of code can improve accuracy and reduce the time required by some performance tools.

The third drawback to these profilers is that it's difficult to use both TrueTime and VTune together when using Visual C++ as your development environment. TrueTime cannot instrument code from Visual C++ with VTune installed because VTune renames certain underlying compile and link programs.

Finally, although both tools display call graphs, we found it difficult at times to ascribe performance timings to specific subsystems. For instance, pathing was sometimes called from both movement and retargeting code, but we were not able to determine which subsystem was using more of the pathing code. TrueTime was generally accurate about this, but in some cases, the numbers it reported just didn't seem to add up. In this type of case, we had to place our own timing code directly into AoK to get complete results.

Regardless of how good today's profiling tools are, they have no understanding of or insight into the underlying program they profile; profiling and analysis tools would be significantly more useful if they had some understanding of what the application was attempting to accomplish. With that kind of additional functionality, these tools could provide performance statistics that would greatly enhance the programmer's ability to improve the application performance. Until that day arrives, you'll have to add profiling and analysis code to your application for even the most mundane performance information beyond simple timings and call graphs.

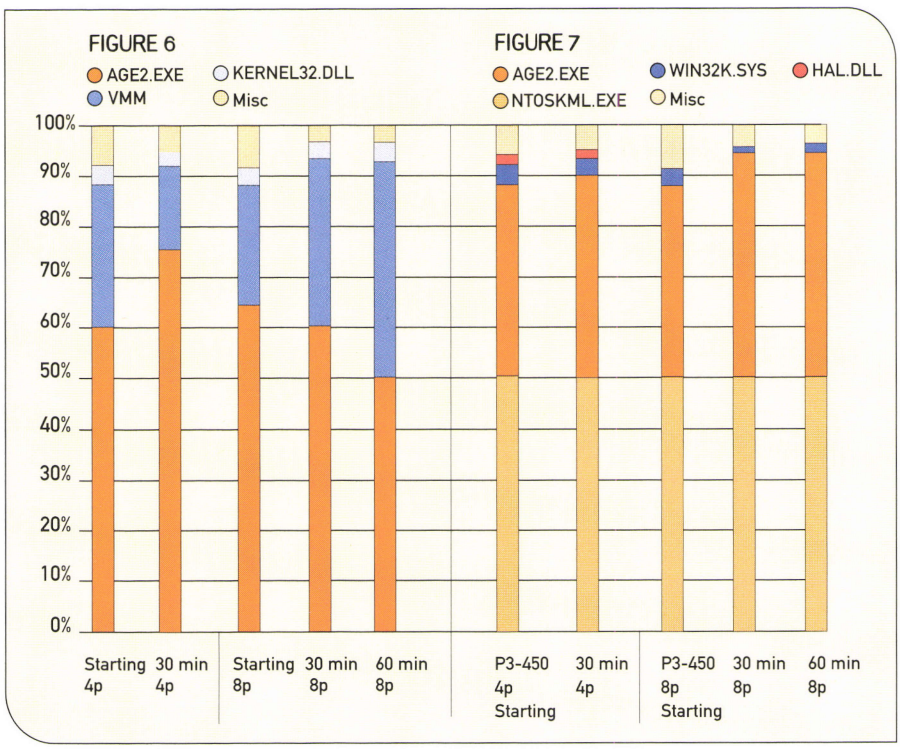


FIGURE 6 (left). Four-player and eight-player game CPU process utilization (Pentium-166).
FIGURE 7 (right). Four-player and eight-player game CPU process utilization (dual Pentium III-450).

Performance on the Minimum System

Since performance statistics can change based on the platform on which the application is running, it was especially critical to get computer systems that matched the minimum system specification. To demonstrate this performance dif-

ferential and the scalability of AoK, two test cases were run on the minimum system configuration and one was run on a regular development workstation (Figure 5). To contrast the data as much as possible in this example, the first test case uses the maximum AoK settings for players (eight) and map size (giant). The second test case conforms to the game settings for the minimum system configuration: four players on a four-player-sized game map.

Using VTune, the percentage of CPU clock cycles spent in each process during an AoK game was calculated for a 60-second period at 30-minute intervals. This was done on the 166MHz Pentium minimum system (Figure 6), and on a dual 450MHz Pentium III development workstation (Figure 7).

As you can see, the four-player game performs well on the 166MHz Pentium. The AoK process starts at approximately 60 percent of the CPU and increases to about 75 percent after 30 minutes. The additional time devoted to the virtual memory manager (VMM) process at startup is caused by AoK data as it is paged in and accessed for the first time. In contrast, the amount of CPU time used by AoK in the eight-player game degrades over time. This is due to the additional memory requirements to support so many players and such a large game map. The CPU reaches the point where it's spending almost as much time managing the virtual memory system as actually executing the game itself.

Since the development workstation (Test PC 2) is a dual-processor system and AoK is single-threaded, the second CPU is idle as the kernel runs. This is why the NTOSKRNL is shown as approximately 50 percent of the CPU.

As both the four- and eight-player games progress, the AoK process continues to use more and more of the CPU. There is no downward pressure being applied from other processes as there was for the 166MHz Pentium for eight players.

If it had not already been established that four players was the number of players to support on the minimum system, these same statistics could have been collected for a varying number of players. Then we could have set the maximum number of players based on how many players could fit within the memory footprint of 32MB.

Our Custom Profiling Tool

To complement and augment the results from the commercial profilers we were using, we developed an in-house profiling tool as well. This tool logged the execution time of high-level game systems and functions (telling us how much time was spent by each one) and told us the absolute amount of time a section of code took to execute — a sanity check for performance optimizations that we sorely needed. Our profiling system consisted of four simple functions that were easily inserted and removed for profiling purposes and relied on a simple preprocessor directive, `_PROFILE`, that compiled the profiling code in or out of the executable. This let us keep our profiling calls in the code, instead of forcing us to add and remove them to create nonprofiled builds. You can download an abbreviated example of the profiling code from the *Game Developer* web site (www.gdmag.com).

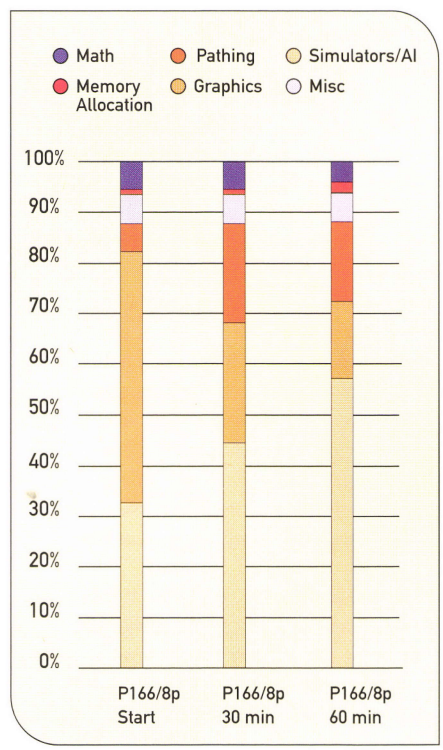


FIGURE 8. An eight-player-game breakdown of internal AoK performance (Pentium-166).

While VTune told us how much of the CPU AoK was using (Figure 6), our custom profiler told us how much time was being spent on each of AoK's major subsystems (Figure 8). This additional information told us interesting things about the performance of AoK and where we might be able to improve performance. You can see in Figure 8 that the amount of time devoted to game world simulation and unit AI increases from approximately 33 percent to approximately 57 percent of the AoK process over the course of three samples at 30-minute intervals during an eight-player game.

Looking back at the process statistics from VTune in Figure 6, you see that the amount of time spent in the VMM increases while the time spent on AoK decreases. Since AoK spends more time in simulation/AI and the operating system spends more time manipulating virtual memory, we can propose some theories to explain this:

- The simulation/AI code is allocating more memory over time without freeing memory, stressing the VMM. However, skipping ahead to Figure 5, we see this probably isn't the case since the memory footprint isn't skyrocketing.
- The simulation/AI code is allocating and deallocating so much memory that as time goes on, the memory heap is becoming fragmented, and that's slowing memory allocation. The only way to confirm this theory is to instrument the code and determine where, when, and how often memory is allocated.
- The data being processed by the simulation/AI is so large or being accessed so randomly that it constantly causes the VMM to flush data from memory and read in new data from the virtual memory swap file.

More data would be required to determine the cause of this problem. It would also be good to break the "simulation/AI" group down into more discrete components for timing.

Our timing code relies on the Assembly instruction `ReadTimestampCounter` (`RDTSC`), but it could also have used the Win32 `QueryPerformanceCounter`, or another fine-grained counter or timer. We chose `RDTSC` because it was simple to use, it works on all Pentium (and later) processors (except

some very early Cyrix Pentium-class parts), and these profiling functions were based on extending existing code.

Finally, although both tools display call graphs, pathing was sometimes called from both movement and retargeting code but we were not able to determine which subsystem was using more of the pathing code. TrueTime was generally accurate about this, but in some cases the numbers it reported just didn't seem to add up. In this type of case, we had to place our own timing code directly into AoK to get complete results.

As I stated earlier, it was difficult to assign performance timings to specific subsystems based on the results of the commercial profilers that we used. To remedy this, we built functionality into our custom profiler to determine how much of each system's time was spent in, say, pathing. Here's how our profiler works. During profiler initialization (`_ProfInit`), the static array of profiling information (`groupList`) is initialized to zero, and the CPU frequency is calculated. The size of the `groupList` array matches the number of profile group entries in the `ProfileGroup` enum list in the `prof` header file. The CPU frequency is calculated with a simple, albeit slow, function called `GetFrequency`. (Alternately, this could have used one of the specific CPU identification libraries available from Intel or AMD, but this code works transparently on Windows 95/98 and NT and across processors without problems.)

The final part of initialization seeds each `groupList` array entry with its parent group. Since the `groupList` array entries match the `ProfileGroup` enums in order, the `ProfileGroup` enum can be used as an index into the `groupList` array to set the parent group value. Using the `SetMajorSection` macro significantly simplifies this code by encapsulating the array offset and assignment. More importantly, it allows us to use the stringizing operator (`#`) to record the parent group's `ProfileGroup` declaration as a string (`const char *`) for use when formatting our output.

The second requirement for our custom profiler was that its profiling code had to be smart enough to make sure that the profiling start (`_ProfStart`) and stop (`_ProfStop`) statements were inserted around a function or group of functions in correct pairings. The `_ProfStop` function first makes sure that profiling was started, and at that point the current time is recorded. This is then used to calculate and store the elapsed time. The number of calls made is incremented, and the starting time is reset to zero. We wanted to avoid the situation where profiling starts multiple times on the same group, or where a `_ProfStop` appears before its corresponding `_ProfStart`. To ensure the correct pairing of profiling statements, in `_ProfStart` a check is made to ensure that the function has not already been called by examining the starting timing value `mqwStart`. The current time is then recorded into `mqwStart` using the `GetTimeStamp` function, a wrapper for `RDTSC`.

In `GetTimeStamp`, it should be noted that the `eax` and `edx` registers are used for returning the current 64-bit timing value as two 32-bit values, which are subsequently shifted and combined. In this case, there is no need to push and pop the scratch registers since the compiler is smart enough to recognize the inline Assembly use. However, if this timing code was encapsulated in a macro, there's the chance that the compiler might not recognize it and it would be necessary to push and pop the registers.

Another issue we confronted with our custom profiling system was the accuracy and resolution of timing available from a system that uses two function calls from the calling code (first to `_ProfStart` and then to `GetTimeStamp`). Since we use this timing code to profile larger subsystems and functions, there will be timing variations due to system factors, such as the execution of other processes by the operating system. If we time significantly smaller portions of code, down to a few lines, it's preferable to inline the `RDTSC` call or use it within a macro.

Using the `RDTSC` as a high-resolution timer can present another problem, too. Note that `RDTSC` is not an instruction that will serialize the execution inside the CPU. In other words, `RDTSC` can be rescheduled just like any other CPU instruction and may actually be executed before, during, or after the block of code you're attempting to time. Using a fencing (serializing) instruction like `CPUID` can solve this.

At the end of the program, the `_ProfSave` function saves the recorded profiling information out to a file. The name of the group, the number of calls, the elapsed time spent in the group, the average time per call, its percentage of its parent group, and the parent group name are listed for each profile group. This output is formatted out using the complicated `proftrace` macro, which once again uses the stringizing operator (`#`) to print out the character version of the profile group followed by its information.

Next month we'll wrap up talking about our profiling tools by discussing the memory instrumentation we created for AoK. Then, we'll take an in-depth look at a number of performance issues facing AoK, including unit movement and pathing, and see how they were addressed.

FOR MORE INFORMATION

Ensemble Studios
www.ensemblestudios.com

Intel VTune and C/C++ Compiler
developer.intel.com/vtune

MicroQuill HeapAgent and SmartHeap
www.microquill.com

NuMega TrueTime
www.numega.com

Performance Analysis and Tuning

Baecker, Ron, Chris DiGiano, and Aaron Marcus. "Software Visualization for Debugging." *Communications of the ACM* (Vol. 40, No. 4): April 1997.

Marselas, Herb. "Advanced Direct3D Performance Analysis." Microsoft Meltdown Proceedings, 1998.

Marselas, Herb. "Don't Starve That CPU! Making the Most of Memory Bandwidth." Game Developers Conference Proceedings, 1999.

Pottinger, Dave. "Coordinated Unit Movement." *Game Developer* (January and February 1999).

Shanley, Tom. *Pentium Pro and Pentium II System Architecture*, 2nd ed. Colorado Springs, Colo.: Mindshare Inc., 1997.

ACKNOWLEDGEMENTS

Creating and optimizing AoK was a team effort. I'd like to thank the AoK team, and specifically the other AoK programmers, for help in getting the details of some of that effort into this article. I'd also like to thank everyone at Ensemble Studios for reviewing this article.

Industry-leading research and analysis for the interactive entertainment industry

What's in store for 3D game technology?

What are my competitors up to?

What are the top game publishers planning for 2001 and beyond?

How will platform issues play into the creation of my next game?

How can I attract and retain talented game developers?

Need Insider Information from Industry Insiders?

CMP Game Media Group Research has developed innovative analysis and data products to meet the needs of the interactive entertainment industry. Our products are backed by the experience and trust from these industry leaders: Game Developer magazine, the Game Developers Conference, Gamasutra.com; and the GAMEXecutive Conference.

What are the revenue breakdowns for game hardware and software?

◗ Game Developer Salary Survey

Recruit, retain and compare employees better with the first professional salary survey for North America. The Game Developer Salary Survey covers all key job titles in the game development industry.*

◗ Publisher Profiles

Assess and compare performances of the largest 30 game publishers in North America, Europe and Japan. Get in-depth analysis operations, financial, developer relations, and marketing strategies.**

What game industry trends will affect my future business?

◗ The US Interactive Electronic Entertainment Market

Keep up with the explosive interactive entertainment market with this expansive report.. This semiannual guide is used by industry leaders as a business planning and reference tool for the executive level.+

◗ The State of Game Technology

Understand the unique issues involved with developing A-title games for today's computer hardware platforms. This critical look at game hardware, platforms and 3D game technologies will help even the most experienced vendors, publishers, developers, and investors.+

*Market Perspectives, Inc. & CMP Game Group Research. Sponsored in part by Virtual Search, Inc.

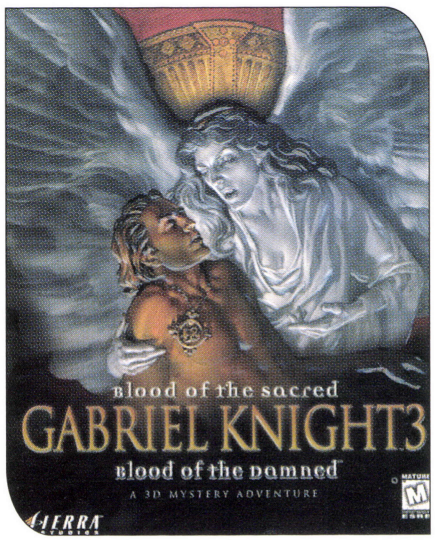
**Produced by Ohmid Rahmat in conjunction with the CMP Game Media Group

+Produced by DFC Intelligence in conjunction with the CMP Game Media Group

CMP Game MEDIA GROUP

www.cmpgame.com/research

Sierra Studios' Gabriel Knight 3



GAME DATA

FULL-TIME DEVELOPERS: More than 45 total, averaged 20 at a time.
CONTRACTORS: 3
BUDGET: Originally well below $2 million, final budget $4.2 million.
LENGTH OF DEVELOPMENT: Almost 3 years.
RELEASE DATE: November 1999 (over a year late).
PLATFORMS: Windows 95, 98, NT, 2000.
HARDWARE USED: Lots of expensive stuff that quickly became obsolete — Dual Pentium Pro 200s, Pentium II 266 up to Pentium III 500. 3D cards by Matrox, Nvidia, ATI, and 3dfx, and whatever else we could get our hands on.
SOFTWARE USED: 3D Studio Max, Character Studio, Visual C++, SourceSafe, CodeWright, VTune, MKS Lex & Yacc, Photoshop, special "sound guy magic."
TECHNOLOGIES: DirectX, Bink Video, Standard Template Library, LZO and zlib (free, open-source compression libraries).
LINES OF CODE: 350,000 lines of C++ (not including original engine), 40,000 lines of script and logic, 8,000 lines of (spoken) dialogue, 36,000 unique resource files, 1,500 average triangles per actor, 185,000 cups of coffee consumed.

Gabriel Knight 3 (GK3) is a traditional "Sierra-style" murder-mystery adventure game that tells its story through a complex, nonlinear mix of dialogue trees, scripted sequences, movies, and puzzles. Most of us should be familiar with this kind of game — it's paced at the speed of the player, and involves a lot of "inspect the monkey" and "use the banana on the monkey" type of interaction to move the story forward. GK3 differs from its predecessors technologically in a variety of ways, but most important is the fact that it moves the genre to full 3D.

GK3 offers a freely roaming camera that lets players go where they please and zoom in on whatever they like. This isn't just a gimmick — this single feature changes the game radically, making it more like an interactive movie and less like an interactive comic book. As we found out, moving from 2D to 3D is not just one-third more work, it's more like three times as much work. It affects nearly all aspects of the game, including both the design and the engine. It's not as simple as just drawing your actors and environments with polygons instead of sprites. Suddenly you have to start worrying about camera angles and dramatic effects that were never possible or necessary in 2D, at least not without resorting to a prerendered movie.

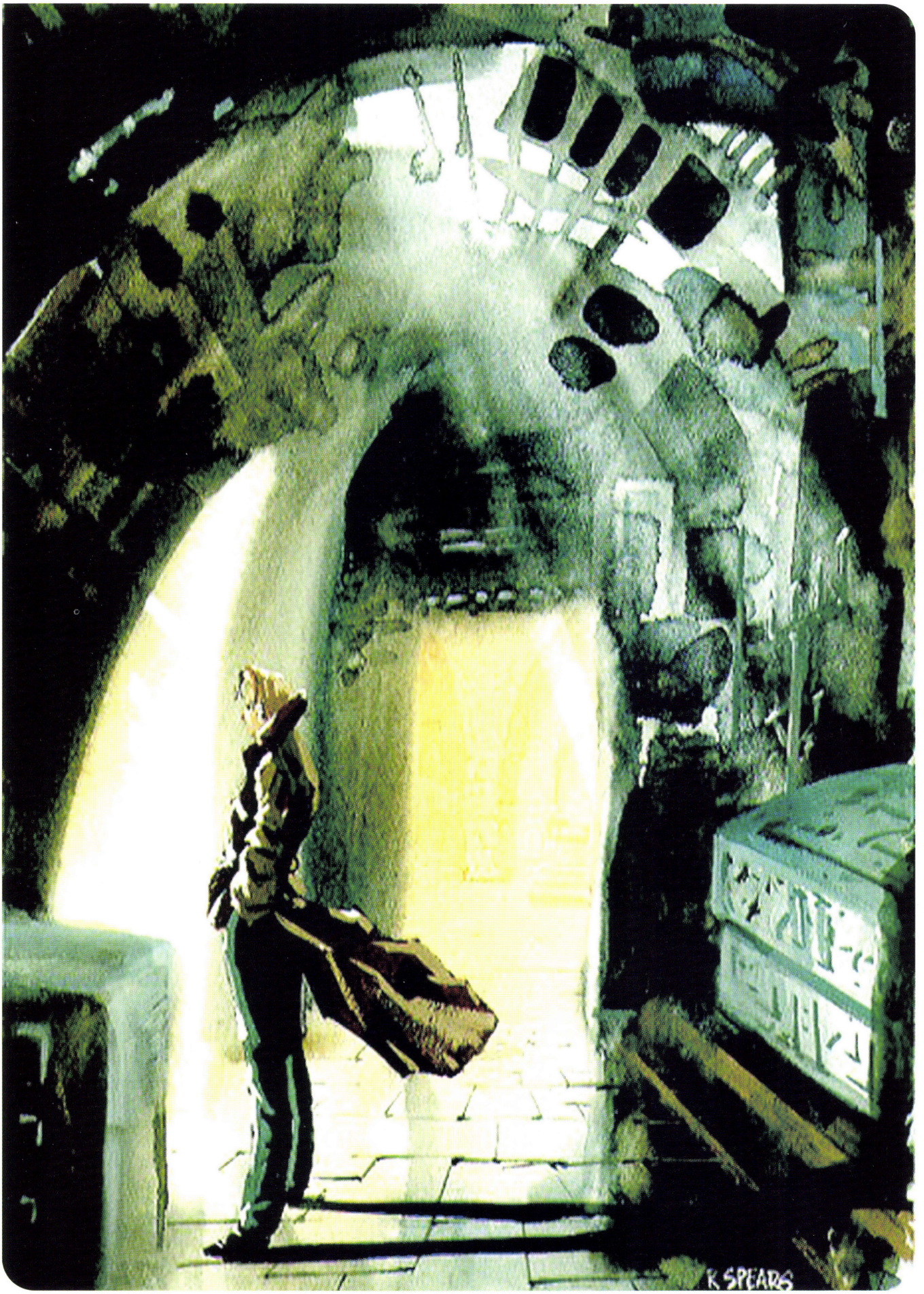
Another part of GK3's design was the ability to give the player the option to turn off cinematic camera cuts during dialogue sequences. The idea was that players could be the director and choose their own camera as the action was unfolding. This had a serious and very expensive effect on the art: it meant that artists could take no shortcuts with their animations. In a prerendered movie, an animator has full control over the camera and can avoid bothering with anything that's outside of its view. This saves a lot of time. In GK3, because players may at almost any time decide to take control of the camera, they would be able to see the action from any angle they pleased and go "behind the curtain" so to speak. Therefore, the animators needed to make sure that the entire scene could be both viewable and good-looking from any angle. This increased their workload by an order of magnitude.

Previous Sierra adventure games, including GK1 and GK2 as well as the Space Quest and Leisure Suit Larry series were built using the "SCI" game engine. SCI was developed and maintained by Sierra Oakhurst and, for a variety of reasons, Sierra Northwest (the division I worked for) decided to stop using it. This single decision probably affected the project more than anything else did — it meant that GK3 was to be the first adventure game Sierra had built completely from scratch in a very long time.

From the start, the project had some important things going in its favor. Sierra hired an experienced engineer, Jim Napier, who got the game started by developing the G-Engine, a 3D rendering, sound, and animation toolkit that provided the low-level foundation for us to build the game upon. After the engine's completion, Jim unfortunately had to leave the project to start on the fledgling SWAT 3 as its lead. The G-Engine was on the whole a successful part of GK3; it provided a stable base for the game and was easy to use and understand. The team was also able to reuse some of the tools and concepts that SCI had provided, such as the content database and lip-synching tools.

Despite these initial advantages, the project faced problems almost from the start. The team had to build a new game

AUTHOR'S BIO | *Scott Bilas was a senior engineer at Sierra Studios and the technical lead on Gabriel Knight 3. He is currently working at Gas Powered Games, where he spends his time inserting needles into haystacks. Scott lives in Seattle and thinks everyone should get out of their cars and onto their feet. He can be reached at scott@gaspowered.com or scottb@aa.net.*


K. SPEARS

engine and most of the related content development tools from scratch, but team members severely underestimated the time, cost, effort, and experience required to construct these tools and establish effective development processes around them. The initial development team was not up to this task.

In the early days of the project, the engineering team must have been living in a magical dream world — I can't find any other way to explain it. When I joined the project in early 1998, GK3 had already been in development for more than a year and a half, and it was scheduled to ship at the end of that summer. I realized that this would never happen because at that point the game was a hacked-up version of a sample application that Jim Napier wrote some time earlier to demonstrate the G-Engine. Sample code being used in a production environment should send shivers up the spine of any experienced engineer. Malignant growths of code were added haphazardly whenever a new feature was required, making the game extremely unstable and difficult to maintain. This problem fed on itself and grew worse over time. One example of this problem was the game's horrendous start-up time. The file and resource system, while sufficient for a sample application's minimal resources, completely fell over when faced with the tens of thousands of files and hundreds of directories in GK3. The game took over a minute just to start up and display the title screen.

Most of the game's non-art content, such as a story sequence involving simple dialogue exchanges between two characters, was initially hard-coded into the game in C++. This was a nightmare for a couple of reasons. First, engineers were creating content instead of working on the engine, and engineers are generally not the best people for creating good content (and they tend to be very slow at it as well). Second, the tiniest changes to the game, such as choosing a different line of dialogue or altering an animation sequence, required recompilation. This made the content development process unbelievably inefficient. Artists would potentially have to wait weeks to see their work integrated into the game. This resulted in engineers resenting artists "chucking art over the fence" and probably inspired similar resentment on the art side.



Hint: Don't walk on that bridge on the left.

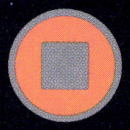
If GK3 was to ship at all, all of this had to change. And so it did.

What Went Right

1. Redesigning the engine. A month after I joined the team, we decided to rebuild the game engine and a little while later I took over as its architect. We spent a couple of weeks in roundtable design sessions with engineering advisors from other projects (including Jim Napier from SWAT 3) and used a low-tech Class-Responsibility-Collaborator (CRC) card design technique to hash out the systems we would need and how they would fit together. I thought all of this went pretty well, though it was slower than most of us liked. Once we started coding, though, things really got moving. The application core was rewritten in a weekend, and then individual systems (user interface, scene abstraction and configuration, font rendering, the console system, and so on) were developed and integrated as fast as we could manage.

In proposing the re-architecture, we gained the full support of upper management, specifically Mark Hood, the general manager of Sierra. They really had no choice, considering that the only other option was to cancel the project, but I think it's important to recognize the risk that they took with us and give them credit for believing in our ability to rebuild the engine. Throughout development, Mark was always 100 percent behind us, and never wavered in his desire for us to ship a triple-A title of the highest quality. Despite our lack of experience, we largely delivered what we set out to accomplish.

Unfortunately, I don't think a lot of the team members outside of engineering ever really understood why we rebuilt the engine. Looking back, I wish we had spent a little more time explaining to them the dire situation it was in. Although it took months to redesign and rebuild the game engine (a time that understandably confused and frustrated everybody), it ultimately improved our team's productivity immensely and made it possible to ship the game. The new engine was stable, flexible, and although it still had architectural problems (due to our inexperience), it worked properly and performed well.



SCORE YOUR OWN GAME.



LicenseMusic.com is the web's leading provider of music for the film, television, advertising and interactive industries worldwide.



From jungle and drum & bass to sweeping orchestrals, we have the soundscape for any scene.



LicenseMusic.com is available 24 hours a day and it is simply the easiest way to search and preview the music you need. To license and instantly download CD-quality audio files directly to your desktop, just log on to LicenseMusic.com.



LicenseMusic.com represents the catalogs of over 175 record labels, music publishers and production libraries. There are over 50,000 pre-cleared tracks available on-line, available instantly for your game.

YOU HAVE THE POWER TO CHOOSE.



LICENSEMUSIC.COM

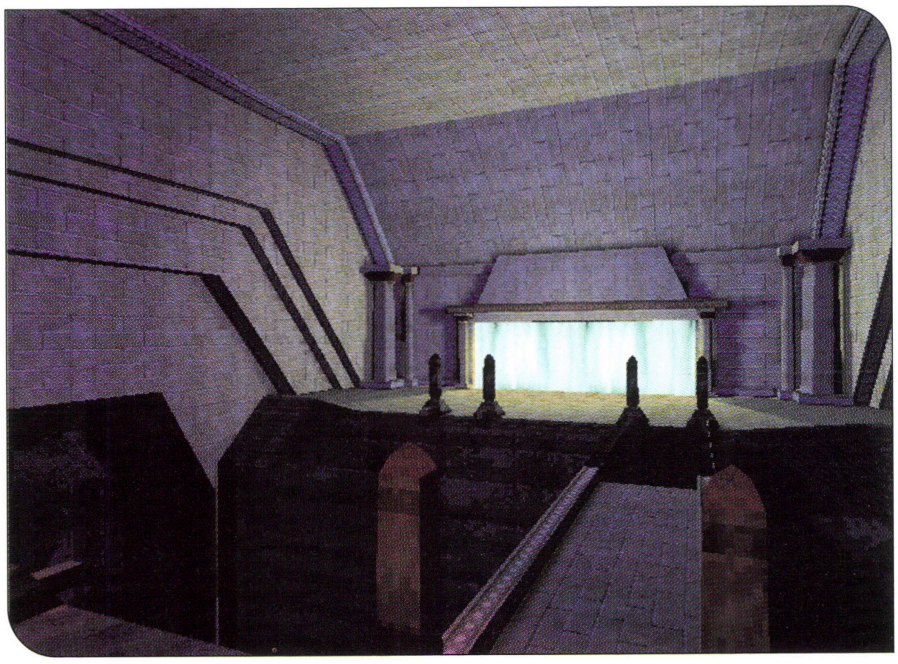
2. Data-driven engine. GK3 is a content-heavy game that ships on three CD-ROMs and includes more than 800MB of compressed non-movie data (consisting largely of texture maps, MP3 dialogue and music, and animations). There are thousands of lines of dialogue and almost as many logic rules tying it all together. During the re-architecture, we went with a data-driven approach, putting as much as we possibly could into text files so that non-engineers could create and maintain content. We were very happy with the results.

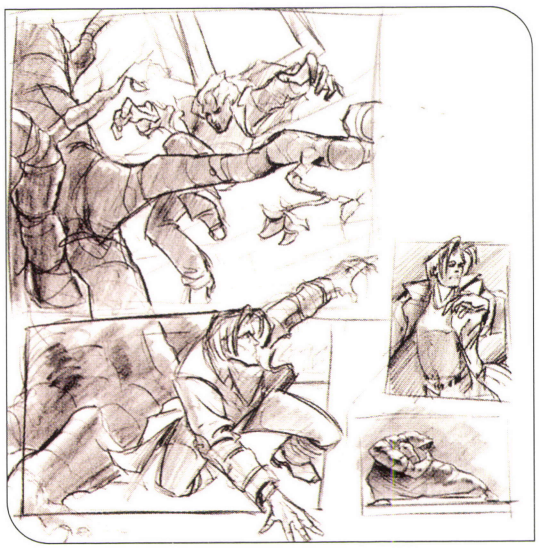
One of the project's major successes in this area was its flexible scripting system, "Sheep," which used a C-like language and was implemented using a simple compiler and p-code interpreter. The compiler was built with our old friends Lex and Yacc from the Unix world. Originally designed just for the game's animated sequences, the Sheep engine ended up being used for custom rules processing, event handling, resource packaging, scene configuration, debugging, the development console, and even for a little bit of testing automation.

I can't stress this enough to developers: Build a scripting engine, even if you don't think you need one. Make it generic enough so that it can be reused in as many places as possible. I think many engineers are scared of building one because they think it will take too long to develop or that it will execute scripts too slowly. This was certainly the case with the original GK3 engineering team. I've found these fears to be completely unfounded — a scripting engine will pay for itself many times over, and can be easily optimized to approach the speed of C++ code. Also, don't invent a new language. Pick a programming language that one of those "Teach Yourself Something Useful in 21 Days" books exists for, and you can buy copies for your scripters if they don't already know the language (although this wasn't necessary for the GK3 scripters). This will save you the time you would have spent documenting syntax and training scripters had you used a completely proprietary language.

Another benefit of Sheep was that when combined with our redesigned file and resource manager, we were able to cut down dramatically the time necessary to integrate art into the game. The old engine used C++ to reference art assets, which meant that artists needed to wait for the next build (at best) before they saw their work in the game. We reduced those days or weeks of waiting down to minutes or hours and almost completely removed the engineers from the picture. Under the new system, artists could check in their work and have one of the scripters integrate and demonstrate it immediately on the existing build.

We added clipboard support too, so that developers could use GK3's console to paste Sheep code directly into the game. The scripters could Alt-Tab away from the game, grab a section of test code from their text editors, Alt-Tab back into the game, and paste it into the console to see immediate results. With these kinds of features in place, content poured into the game at a blinding pace.





TOP: There's a big ugly demon beyond that veil.
BOTTOM: Hand-drawn concept art.

3. The design. The game's design was a major success and deserves special mention. GK3 would have simply fallen over and died had we had a less experienced designer than Jane Jensen. Throughout the entire development process, the one thing that we could count on was the game design. It was well thought out and researched, and had an entertaining and engrossing story. Best of all, Jane got it right well in advance — aside from some of the puzzles, nothing really needed to be reworked during development. She delivered the design on time and maintained it meticulously as the project went on.

DIGIMATION

DIGIMATION DELIVERS REAL-TIME 3-D GRAPHICS LIBRARIES FOR GAME DEVELOPERS

Real-Time 3-D Libraries Incorporate Intel® Scalable 3D Graphics Allowing Reduced Game Development Time and Costs.



Digimation today announced it will distribute five real-time (RT) libraries, featuring technology developed by Intel Corporation, which will enable developers to create stunning real-time, scalable 3-D graphics while reducing development time and costs. These libraries not only enable games to look great on existing hardware but to adapt to tomorrow's more powerful computing environment by producing even more compelling images. "The Intel Multi-Resolution Mesh (MRM) technology saved us an enormous amount of time and money when developing Dark Reign* 2," said Josh Resnick, President of Pandemic Studios. "We were able to put those energies and resources into areas that had more impact for users. The only thing I'd love, is for the tools not to be shared with my competition." The libraries, based on Intel technology, consist of: MultiRes 2 RT, SubDiv RT, Animate RT, Toon RT, and Particle RT.

The first of the new libraries, **MultiRes 2 RT**, is the latest version of last year's MRM technology which received "Five out of Five Stars" from Game Developer Magazine November 1999. MultiRes 2 RT allows programmers and artists to create scalable 3-D models that automatically adjust resolution according to distance, target frame rates, and available processing power. Leading game companies already using the Intel MRM technology include Valve Software, Pandemic Studios, DreamWorks Interactive and Sunstorm.

SubDiv RT provides users with a method of increasing the polygon count of low-resolution 3-D models by adding polygons in real-time. This effectively means a user can create and initially deliver a low polygon count model that is compact in size and then dynamically increase the resolution when it is displayed, providing a richer visual experience than the smaller sized model.

Animate RT allows content authors to animate objects by manipulating embedded skeletal systems. Animation can be pre-authored with other software packages such as Character Studio, or Bones Pro 2 inside of 3D Studio MAX,, or generated in real-time by the Inverse Kinematic (IK) technology that is included.

Toon RT lets users transform and render 3-D models in a variety of non-photorealistic methods. Some of the "looks" that Toon RT can perform include: cartoon, pen and ink, crayon, black and white and solid line silhouette outlining.

Particle RT gives content creators a complete particle system for generating thousands of particles in real-time that disperse realistically and are affected by a variety of environmental conditions such as wind, gravity, walls and positive and negative magnetic fields. The particles can be scaled to different sizes dynamically and bitmaps can be added to achieve widely differing effects from billowing smoke to fiery explosions.

DIGIMATION
The Digital Animation Company

WITH intel Optimizers™ SCALABLE 3D GRAPHICS

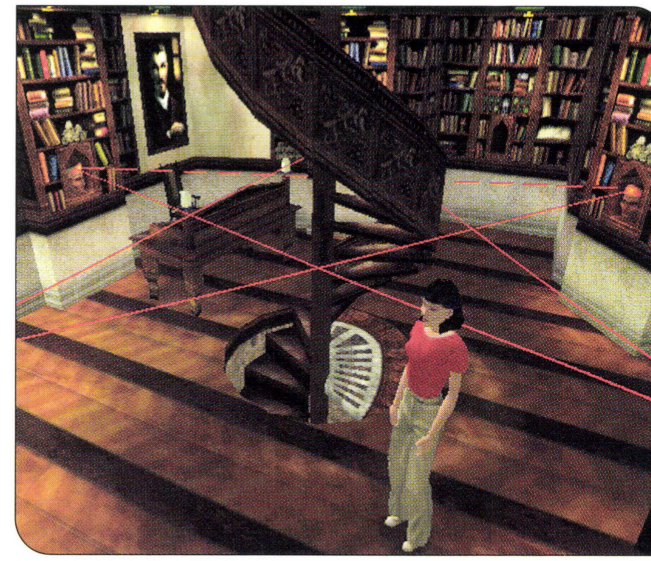
Digimation develops and distributes plug-in technology for 3-D software as well as stand-alone graphic oriented programs for the PC. Digimation programs are sold through a network of worldwide dealers and distributors as well as direct. Visit Digimation on the World Wide Web at www.digimation.com, or contact the company directly at 800-854-4496 or 504-468-7898.

Digimation, the Digimation logo, and Maximizers are registered trademarks of Digimation, Inc. Autodesk, the Autodesk logo, and 3D Studio are registered trademarks, and 3D Studio MAX and Discreet are trademarks, of Autodesk, Inc.

*Third party marks and brands are property of their respective holders.

4. The audio. Audio designers and engineers rarely get the credit they deserve and often end up taking second place to the people drawing the pretty triangles. But GK3 is an adventure game, and as such it lives and breathes on the ability of its dialogue and supporting audio to immerse the player in the story. Many reviewers picked up on the great audio they found in GK3, often rating it as one of the best parts of the game (that is, those reviewers who didn't have a silly personal issue with Tim Curry cast as Gabriel Knight). From a development point of view, audio content was something we could always rely on. David Henry, GK3's composer and lead sound designer, had it all done long before we actually needed it, and was therefore able to spend time polishing the audio and adding lots of small details to it. And in stark contrast to the other parts of the game, integration and maintenance of the audio content went as smooth as glass.

5. A talented, dedicated core of developers. GK3 never would have shipped without the heroic efforts of critical developers in key places across the board — art, scripting and logic, engineering, design, and testing. These people took over various parts of the project on their own initiative and kept pushing until things were done and done right. The loss of any one of these individuals would have severely crippled the game's chance of making it out in 1999, if at all. Among the crowd, two names deserve special mention. Halfway through the project, we picked up Jessica Tams as our content lead, who took over the content and put it in order. She wrote nearly all the scripts and logic for the entire game, completed them on schedule, and somehow made them all work despite the problems with the engine (more on these problems in a moment). Lead animator Ray Bornstein came onto the project with a year left to go, put the animations in order, created a realistic schedule, and made the animators stick to it.



Construct a familiar symbol to open the stairs into the floor in one of the game's cooler sequences.

What Went Wrong

1. Team casting problems. When someone is placed in a role in which they don't belong, I call this being "badly cast." Many of the problems with GK3 resulted from developers being badly cast in their roles, usually because the project requirements were so severely underestimated. To give you an idea of the casting problems we had, consider this: we went through a total of two producers, three art directors (we spent the last year of the project without one), and three project leads (the producer was forced to take over as project lead towards the end).

This was an ambitious, massive project that required experienced engineers and the original team was simply not up to this task. GK3 was initially built from members of the SHIVERS 2 team (one of the last games built with SCI) and they had practically no 3D experience. Engineers under the venerable SCI engine were basically scripters and putting them in charge of building a game engine from scratch was like feeding them into a furnace. To make things worse, the developers that were in over their heads didn't ask for help, which gave management a false sense of progress.

2. Severe morale problems. Hundreds of books and articles have been written about this and here we have yet another Postmortem listing it under "what went wrong." It's time for me to get on my soapbox. To managers everywhere: morale is

PROGRAM FOR YOUR FUTURE!

Earn your B.S. or M.S. in Computer Science through distance education.

- Prepare for one of the thousands of computer science jobs available
- Increase your earning power
- Study from your home or office at your convenience
- Approved by more than 330 companies
- Follows ACM/IEEE guidelines
- B.S. in Information Systems also available

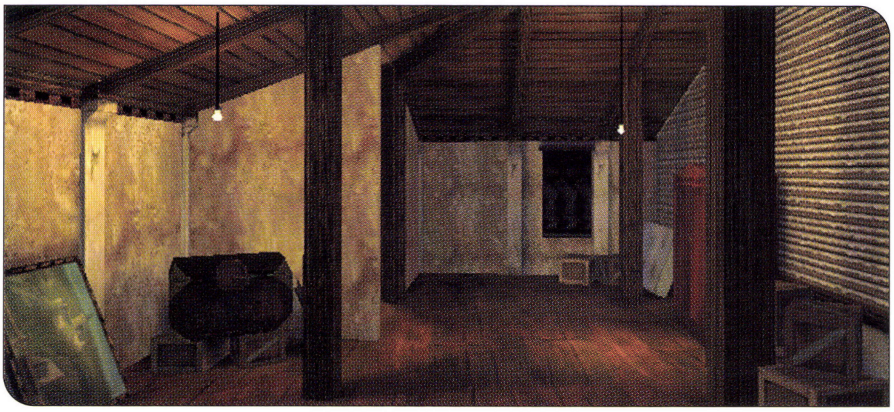
DETC ACCREDITED MEMBER

ACCREDITED MEMBER DETC

For a free catalog call 1-800-767-2427 or visit *cs.aics.edu*

Birmingham, AL

AMERICAN INSTITUTE FOR COMPUTER SCIENCES



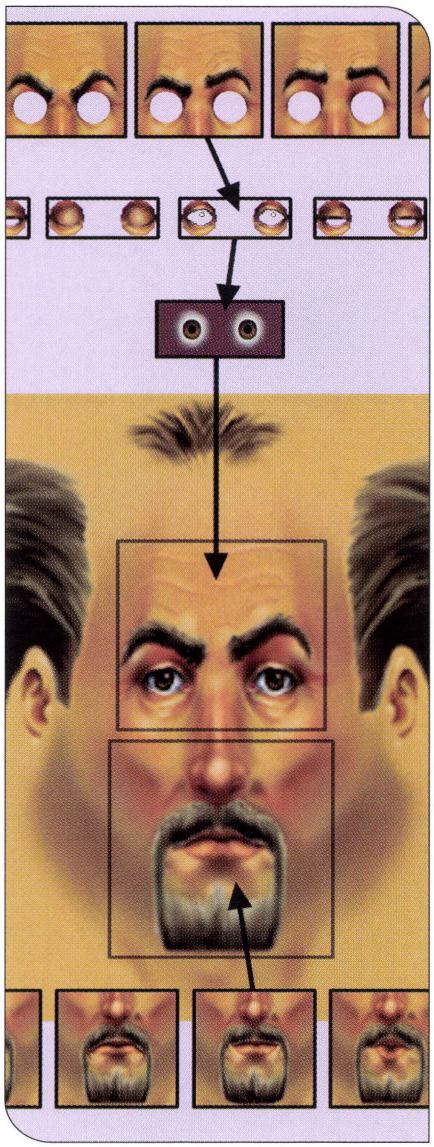
one of those icky personal political things that many of you avoid dealing with. But you need to understand that your development team is not a factory churning out content and code. To paraphrase Peter Sellers in *Being There*, "The team is a garden of creativity that requires regular watering and sunshine in order to build strong roots." Loyalty is not something that comes easily. The job market is very competitive — your best developers will simply leave and work for somebody else if they aren't treated well and maintained properly. On GK3, there was a serious lack of love and appreciation throughout the project. Recognition of work (other than relief upon its completion) was very rare, lacked sincerity, and was always too little, too late.

Internally, a lot of the team believed that the game was of poor quality. And of course, the many web sites and magazines that proclaimed "adventure games are dead" only made things worse. Tim Schafer's GRIM FANDANGO, although a fabulous game and critically acclaimed, was supposedly (we heard) performing poorly in the marketplace. Rumors circulated among our team that GK3 was going to lose money, due largely to our high burn rate.

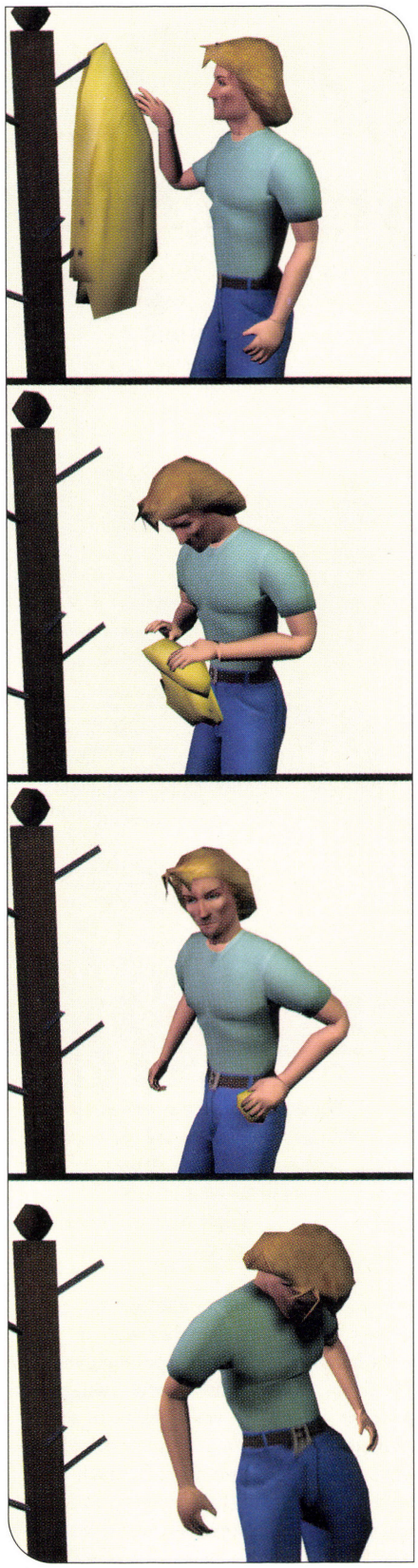
The low morale resulted in a lot of send-off lunches for developers seeking greener pastures. GK3 had a ridiculous amount of turnover that never would have been necessary had these people been properly cast or well treated in the first place. More than 45 developers worked on GK3 (the average standing team size was 15 to 20), and now, just a few months after it shipped, only seven remain at Sierra. Strangely, the opposite also happened — several of our developers were included in Sierra's mid-1999 housecleaning layoffs but these individuals were allowed to stay on for a couple of months, postponing their last day until we shipped GK3. I believe this was done in good faith out of respect for the developers' hard work up to that point but it ended up being a prolonged drain on morale. Having a small group of people who are (understandably) upset with your company for laying them off and actively looking for a job while still trying to be productive and contribute to a project is a tough situation that should be avoided.

After a certain amount of time on a project like this, morale can sink so low that the team develops an incredible amount of passive resistance to any kind of change. Developers can get so tired of the project and build up such hatred for it that they avoid doing anything that could possibly make it ship later. This was a terrible problem during the last half of the GK3 development cycle and as a result there are many aspects of the game that we aren't proud of. These were problems that should have been fixed but nobody wanted to take the time to correct them because we were so focused on trying to get the game out. I don't think anyone on the team is directly at fault for this and I don't know what we could have done to correct this problem.

3. Schedule problems. Our engineers never had an accurate development schedule — the schedules we had were so obviously wrong that everybody on the team knew there was no way to meet them. Our leads often lied to management about progress, tasks, and estimates, and I believe this was because they were in over their heads and weren't responding well to the stress. Consequently, upper management thought the project was going to be stable and ready to ship long before it actually was, and we faced prolonged crunch times to deliver promised functionality. More frequent and honest communication within the team would have avoided a lot of this.



ABOVE: Dynamic texture composition used to generate facial expressions and eye movements: eyebrows + eyelids + eyeballs + mouth equals expressive lip-synched face.
TOP LEFT: Poke around in here and you may annoy an arch villain.





ABOVE: 2D interfaces are overlaid on the 3D scene to keep the player immersed.
LEFT: Just about all of Gabriel's inventory fits in his pants.

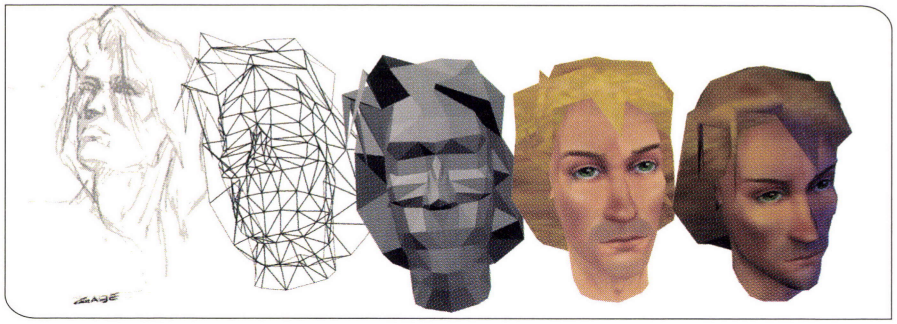
GK3 had few real milestones, which undermined our ability to track progress. There were some milestones very early on in development, but the focus on shoveling visible features into the game turned them into worthless smoke-and-mirrors demos. The concept of milestones eventually was discarded and was replaced with two simple and unofficial goals — beta and release to manufacturing. In the push to ship the game, we simply forgot about milestones (because "we're almost there!"), put the blinders on, and worked like mad.

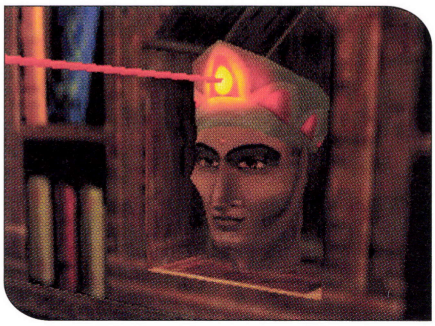
Crunch mode is a reality on most projects, but you should not gear up for one unless the light at the end of the tunnel is really in sight. The GK3 team was pushed into crunch mode three separate times, each time thinking that we were almost ready to ship. Most of the last year of the project we spent in this mode, which meant that even small breaks for vacations, attending conferences, and often even taking off nights and weekends were looked down upon. It was time that the team "could not afford to lose." The irony is that this overtime didn't help anyway — the project didn't move any faster or go out any sooner. The lack of respect for our personal lives and attention to our well-being caused our morale to sink.

Because of the high turnover, GK3 always had a high percentage of developers new to the team. Faced with mandatory overtime, these new people understandably felt it was unfair to be "punished" by paying for problems caused by the original team or things that they felt management had brought upon itself. GK3 became a black hole that sucked in many developers from other projects, often at the expense of those projects. Artists were shifted off the team to cut the burn rate, and then pulled back on later because there was so much work left to do. Our art team finally got on track in the last year of the project (thanks to Ray), but engineering never got a solid schedule together and as a result we were nearly always late in our feature delivery.

4. Engineering problems. When we rebuilt the game engine, we tried to retain as much of the original code as we could to get the game up and running again as soon as possible. With the exception of the G-Engine, this was a big mistake. Nearly every one of the systems that we kept caused us problems — they were badly designed, buggy, inflexible, and should have been redesigned. Some of these systems never worked correctly throughout the lifetime of the project and had to be hacked around by the content developers to get the game to ship.

Specifically, we had serious problems with the "fidget" system (used by character models when idle or when involved in dialogue), the character model's walker, the vertex animation system, and the conversation and dialogue systems. All of these failed regularly and were regularly "fixed," but each bug fix introduced new bugs, usually in the form of hidden time bombs. The engineers responsible for these systems became very defensive about the problems with them, and usually ended up blaming artists and scripters or even other engineers for the cause. Management, thinking that it would save time, often encouraged content developers to hack and work around the problems rather than fix them properly. We should have ripped these parts of the game out





ABOVE: Gabriel Knight, from concept art to dashing real-time 3D model.
ABOVE RIGHT: Rendering lasers required a little bit of custom code.

and rebuilt them, rather than continually attempting to work around a flawed legacy design. It would have saved a lot of time and hard feelings.

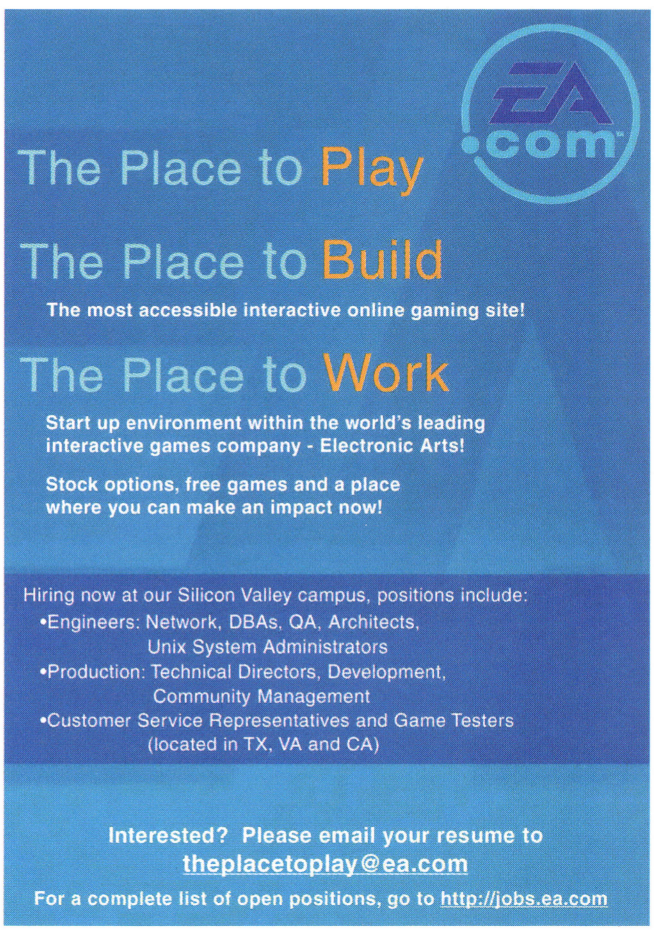
We also faced a lot of problems that were out of our control. Most notable were the technical difficulties with the DirectX drivers provided by hardware vendors for their 3D graphics cards and sound cards, but this probably isn't news to any 3D game developers. These problems were generally features that were implemented improperly or inconsistently, or just outright bugs that caused system crashes or hangs.

We also wasted a few weeks trying to add copy protection. During the final push to ship, we repeatedly attempted to make Macrovision's SafeDisc product work with GK3. SafeDisc has a set of special (and we felt completely unnecessary) antihacking measures that got in the way of the game's execution. It heavily affected performance, dropping the frame rate to a third of its original speed and adding strange intermittent freezes of several seconds while the camera was moving. After getting nowhere with Macrovision's engineering department, we decided to ditch SafeDisc and roll our own (which took less than a day to do). This entire process wasted several weeks of our time and frustrated us all the more because, apart from this one remaining task, we were ready to ship the game. Lesson learned: If you are required to use copy protection, don't put it off until the last month, especially if it's SafeDisc. We weren't the first game to have severe problems working with SafeDisc and probably won't be the last, so if you're using this product be sure to do your homework and try it out well in advance of your ship date.

5. Art and (more) engineering problems. One of the most expensive mistakes a team can make is ramping up art before the engine is ready. This often happens at large game companies because developers need a place to go after they've shipped their most recent game. Unfortunately, this was a serious problem with GK3 — artists were brought onto the game while the original engine was in development, and long before a stable engine was available. They created content for an animation engine that was untested and in doing so built up enough inertia that we ended up having to keep the design. Later we discovered that the engine's design was seriously flawed.

GK3's animation system is vertex-based, meaning that a model's individual vertices are animated. Even using some creative compression methods, this is very expensive in terms of memory usage. Contrast this method with a typical skinned skeletal animation system, which only requires that the bones be animated. The worst thing about this system was not the memory usage, however. It was the impact on content creation, and the repercussions of this requirement were not fully realized until the art team was ramped up and churning out models and animations.

GK3 animations are completely coupled to the meshes of the models that they affect. Once an animation is exported from 3D



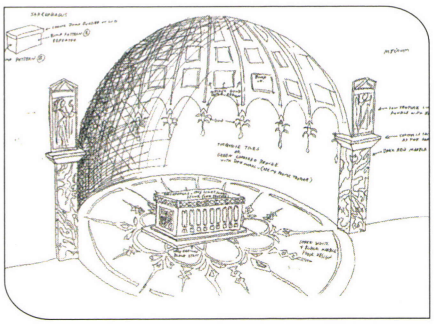
Studio Max, the models it involves cannot be changed in any way, otherwise existing animations created from the old versions of those models would break. Vertices can't be added or removed, and texture maps can only be tweaked, not remapped. Changing a model required re-exporting every single animation that affected the model, which was very time consuming, tedious, and repetitive for animators, and was often impossible to boot. The source assets (the original Max files) had a way of getting lost and usually ended up stored on backup disks as artists left the project. The end result was that once a model was created, it could never be changed. Consequently, GK3 shipped with a lot of bad art that the team was dissatisfied with but had to use. An example of this was the Mosely character (sometimes not-so-fondly called "T-Rex man" internally), whose arms were way too short. Just seeing this guy in the game hurt our morale, and made a lot of us feel poorly about the game. The art was bad and there was nothing we could do about it. Nearly every attempt to change a model was met with the response, "That will require re-exporting all of its animations, which will take too long."

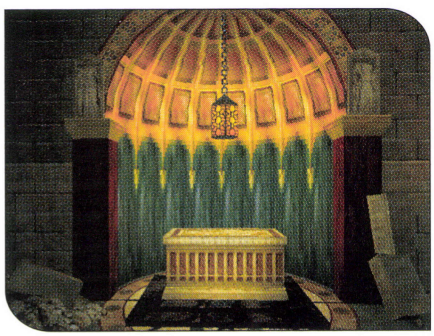
What we should have done was just stop the presses and fix the art. When we first recognized the severe problems with the vertex-based animation system, we should have rebuilt that part of the G-Engine and replaced it with a simple skeletal animation system (as the SWAT 3 team later did on their project). We should have also thrown out all of our existing art and recreated it. In retrospect, that would have saved us a lot of time, given us better performance, and made development proceed more smoothly.

Out The Door

GABRIEL KNIGHT 3 was an extremely ambitious project combined with an extremely inexperienced team. With all that went wrong, you would have every reason to believe it would never have been finished. Despite all that was messed up with our development process however, we somehow managed to get GK3 out the door and ship it in time for Christmas 1999. Best of all, the game works and it works well, against all odds. This is a tribute to the testing skills of our QA lead, Matt Julich, and his team. Though severely understaffed, they did a great job ensuring that the game we shipped was of high quality. There are no crashes and no hangs in GK3 — Sierra will not be issuing a patch.

This isn't to say that GK3 has no problems, just no problems that we had any real control over as developers. To the best of our knowledge, every single critical problem with the game is caused by either bad hardware drivers (sound or 3D card) which can be fixed by upgrading them, or CD-ROM problems due to our oversized CD copy protection. Also, the first disk that shipped in some retail boxes was difficult to read on some CD-ROMs, apparently due to duplication problems. This can also be worked around, as the same file exists on the second disk and can be copied to the hard drive.





Concept and final environment for the end game.

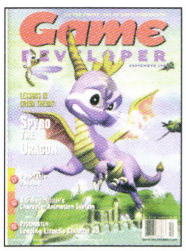
GK3 is certainly not the game it could have been had we been able to knock down a couple of those "what went wrong" issues early on, but it is a high-quality, entertaining game that has been well received by adventure gamers. And that's something we can all be very proud of.





CALL FOR WRITERS

Dear Professional Game Developer,



If you're technically astute and have a way with words, *Game Developer* magazine needs you. We're looking for feature articles on programming (graphics, AI, networking, and so on), art and animation techniques, game design, audio technology, testing and QA issues, and other relevant technical topics. We want to cover topics for variety of platforms, including the PC, console, arcade, and handhelds. Give back to the community some of the hard-won knowledge that you've learned!

We're also on the prowl for postmortems of recently completed games, always looking for people who are interested in reviewing game development tools, and constantly hunting down rabid "Soapbox" column writers.

Please send your article abstract, outline, or wacky idea to: adunne@sirius.com

Writer's guidelines can be downloaded from our web site at: www.gdmag.com/writguid.htm

Thanks,

Alex Dunne, Editorial Director

CMP Game DEVELOPER

VETERAN GAMERS
X-BOX, PS2 & PC TITLES

Cool Publisher Clients
- •EA Sports
- •Mattel
- •Lego
- •AOL

Cool Games
- •NASCAR 2000
- •Pool of Radiance II
- •Madden Football
- •Star Trek: DS9
- •On-line Titles

Cool Place
- •Fun environment with awesome, creative talent
- •Located in scenic Marin, just north of San Francisco
- •Strong project planning focus

Projects to staff
- •X-Box Action Launch Title
- •RPG
- •PS2 Extreme Sports
- •PC & Dot-Com (Driving Sim-Game with Net Based Elements)

"Are You Cool Enough For These Openings?"
- •Senior or Lead X-Box & PS2 Programmers
- •Senior or Lead PC Programmers
- •Senior Graphics Programmer
- •Audio Programmer
- •Physics & AI Programmer
- •Entry Programmers
- •Producer or Associate Producer
- •Art Director
- •Lead 3D Animator
- •3D Animator
- •Lead Artist
- •2D/3D Artist

STORMFRONT STUDIOS

Résumé: Stormfront Studios, Attn: M. Daglow
4040 Civic Center Drive, San Rafael, CA 94903
Fax: 415-461-3865, E-mail: mdaglow@earthlink.net
www.stormfrontstudios.com

ENSEMBLE STUDIOS

10440 N. Central Expwy. Suite 1600 Dallas, TX 75231
214.378.6868 phone 214.378.6464 fax
www.ensemblestudios.com

employment opportunities

ENSEMBLE STUDIOS

ART
- 2D texture
- 3D animator
- low poly modeler
- cinematic
- concept

DESIGN
- user interface
- artificial intelligence
- content

PROGRAMMING
- communications
- 3D console
- 3D graphics
- artificial intelligence

PRODUCTION
- producer
- quality assurance lead



CAPCOM®



CAPCOM IS NOW EXPANDING IN THE US.

We are looking for the very best Programmers, Artists and Animators.

Great Projects!

CAPCOM is well known for its high action, high quality AAA titles (we don't produce anything else). Our focus remains Great Art, Design and Technical Innovation.

Great Location!

Located at the heart of Silicon Valley in Sunnyvale, California USA.

Great Teams!

Join our experienced and productive teams. If you know anyone who works here just ask them what its like, if not just contact us and we can put you in touch with them.

Great Management!

The studio management is drawn from talented and award winning designers, artists and programmers, each with extensive history in the gaming industry.

Great Future!

Come join our team and work on the best titles on tomorrow's platforms.

All inquiries are kept strictly confidential. We are an equal opportunity employer offering excellent salary, incentives and benefits. Visit our website at: **http://www.capcom.com/jobs**
Or contact us at: **resumes@capcom.com**
mail to: **resumes@capcom.com**

Tapes and portfolios can be sent to:
Attn: Resumes CAPCOM
475 Oakmead Parkway
Sunnyvale, California 94086
*Note: Please send copies only as we can not return materials

CAPCOM®

© CAPCOM CO., LTD. 2000 ALL RIGHTS RESERVED. CAPCOM and the CAPCOM logo are registered trademarks of CAPCOM CO., LTD. MAXIMO is a trademark of CAPCOM CO., LTD.

"As a contractor, I use dice.com and dice.com only to keep myself employed. I have made over a quarter of a million dollars in the last 18 months, thanks to dice.com!"

high tech jobs www.dice.com

AN EARTHWEB SERVICE

dice.com
High tech jobs online

Forgive our enthusiasm, but we're PSYCHED

Available Positions

_Design

3D Modelers
Web Developers
Producers
Content Engineers (TCL / Tk exp)
VP Marketing
Chief Designer - Computer Graphics

_Engineering

3D Workspace / Human Factors
C++ Engineer / Software Engineers
Q.A. Engineer / Manager
Tech Writers
Unix Sys. Admin
I.T. Director

- Do you think the Internet could do with a facelift?
- Do you suffer from Windows rage?
- Do you want your Internet experience to feel as good as the games you play?

We do. Here at Muse Communications, we are creating a platform that enables the creation and delivery of applications and Internet content that will make YOU excited. You're gonna dig this...big time.

_muse communications corporation

1950 elkhorn court, san mateo, ca 94403
fax: (650) 378 - 6339 jobs@musecorp.com
www.musecorp.com

INTERACT

WHERE TALENT MEETS OPPORTUNITY

Sometimes it's **what** you know.

Sometimes it's **who** you know.

Interact - where **what** you know and **who** you know come together.

Interact was established in 1993 and is the video game industry's leading recruiting company, specializing in the placement of game programmers, artists, animators, game designers and game producers nationwide. Interact also represents independent studios and "packages" development teams with high profile licensed properties.

Put our resources to work for you.

831 South Douglas St., Suite 119
El Segundo, CA 90245
Tel: 310-643-4700 Fax: 310-643-4750
email: info@interactjobs.com www.interactjobs.com

WE MAKE GAMES FOR THE ARCADE, FOR THE STREET, FOR THE HOME, AND EVEN FOR THE CASINO...



IF YOU WANNA MAKE GAMES, YOU SHOULD BE TALKING TO US.



IMMEDIATE OPPORTUNITIES for ANIMATORS & PROGRAMMERS

www.gamerefuge.com

GAME REFUGE



Merit Industries & Midway Mfg. Images Used by Permission All Rights Reserved

CREATIVECAREERS

gameplay is God.
join the disciples.



Programmers

Senior programmer
* Wide knowledge of C/C++.
* Experience of Windows and/or console programming.
* An ability and good grasp of advanced maths.
* Three years or more of industry experience.
* Good communication skills.

Programmer
* Good knowledge of C/C++.
* Experience of Windows and/or console programming.
* Good understanding of maths.

Java Programmer
* Good knowledge of Java programming
* Knowledge of system, network and database programming
* Good understanding of maths

UDS is a Swedish based game developer currently employing over 40 people and working with world leading publishers. The company has been established for over five years and has strong financial backing. Our aim is simply to be the best.

Applicants should have experience of working in both small and large teams, are motivated, creative and take pride in their work. And of course you love playing games.

Send your CVs and work examples to:

UDS,
Att: Personnel Dep.
Vastgotegatan 13,
S-60221 Norrkoping,
Sweden.

or mail them to jobs@uds.se.

UDS
http://www.uds.se

Artists

Senior Artist
* Extensive knowledge of any major 3D package.
* Softimage experience is ideal but not essential.
* Good knowledge of Photoshop or similar texture package.
* Good knowledge of low polygon 3D modelling & texturing.
* Good knowledge of high polygon/nurbs 3D modelling & texturing.
* Strong character design/animation skills required.
* Three years or more of industry experience.

Graphics Artist
* Good knowledge of any major 3D package.
* Experience of Photoshop or similar texture package.
* Experience of low polygon 3D modelling & texturing.
* Experience of high polygon/nurbs modelling & texturing.



RETRO STUDIOS™

www.retrostudios.com

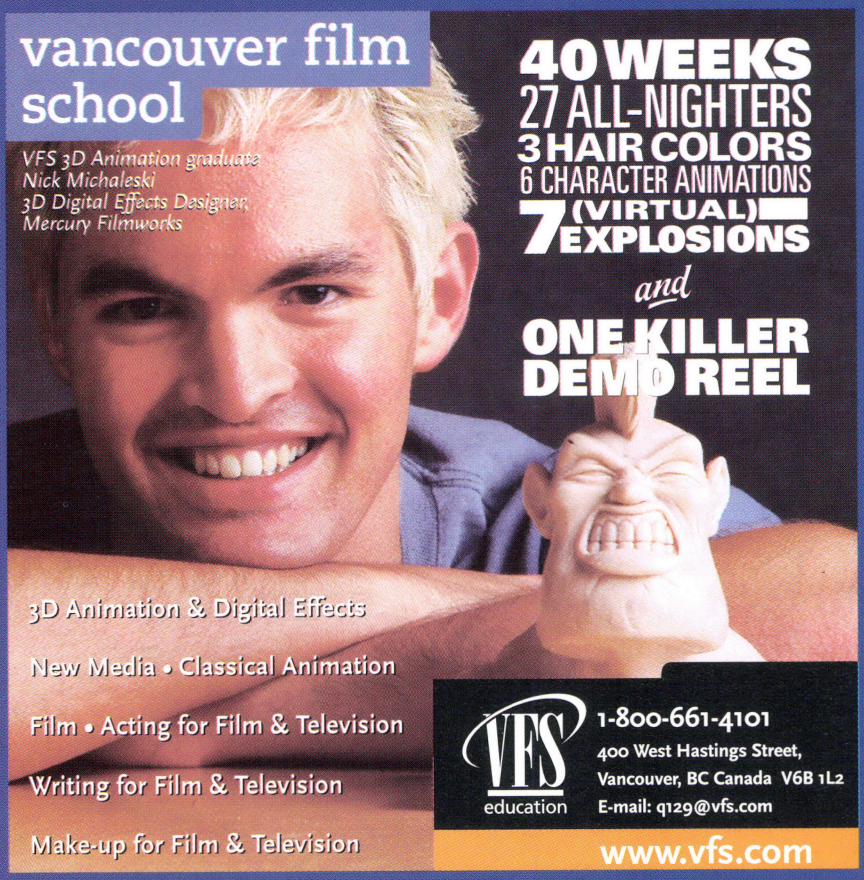
Nintendo®

"HIRING THE SHARPEST MINDS TO DEVELOP FOR NINTENDO'S NEXT GENERATION SYSTEM"

CREATIVECAREERS

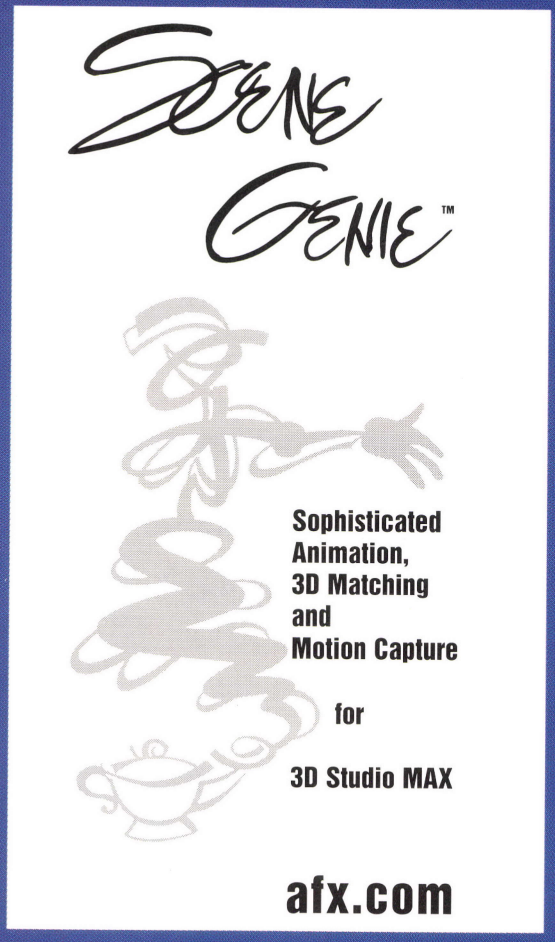


TARGETPAVILION





TARGETPAVILION



To advertise in Creative Careers or Target Pavilion, contact Morgan Browning, 415-905-2788 mbrowning@cmp.com or Jennifer Orvik, 415-905-2156 jorvik@cmp.com



Have you checked out the new Gamasutra.com lately?

Gamasutra.com

Find out why Gamasutra.com is a powerful resource for game developers.

More original game development articles. Gamasutra.com showcases technical articles for information about current trends, essays about game design philosophy, or advice for career management. We have hundreds of helpful articles by game developers for game developers right at your fingertips—original content and reprinted editorial from Game Developer Magazine. Gamasutra.com features articles from developers working at some of the leading game development studios—from Activision to Zombie.

Daily industry news.

Keeping up with our fast paced industry is easy with Gamasutra.com. Track personnel changes, company acquisitions and earnings, product and technology announcements, the best-selling games, and more! It's all available on Gamasutra.com.

Access company & product databases, industry job postings, and discussion forums.

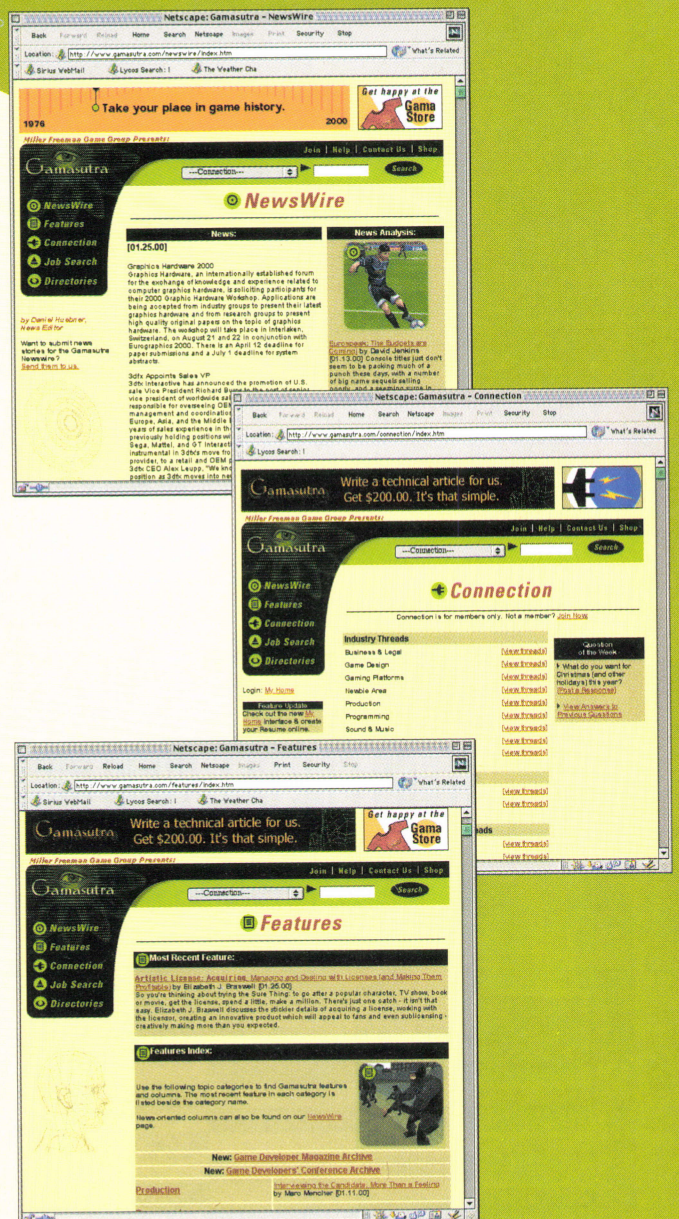
As a Gamasutra.com member, you can search our vast company and product databases to help you accomplish your job. Daily updated job postings will help you find your next interesting project. Our discussion forums enable you to exchange ideas and opinions with developers anywhere in the world.

Unparalleled Game Developer's Conference 2000 coverage.

Before, during, and after the GDC 2000, Gamasutra.com covers the event from the game developer's perspective like no other site. Gamasutra.com provides daily coverage of the show including information about new products, show happenings, and the comprehensive postmortems tailored to the needs of programmers, artists, sound designers, and other fields. Plus, you can access the notes from the GDC 2000 speakers.

60K game developers and growing.

Our registered global membership for Gamasutra.com just surpassed the 60,000 mark! We now have members from countries as diverse as India, Ukraine, Poland, Brazil, Australia, and South Africa.



Free access and Membership.

While Gamasutra.com continues to bring innovation and evolution to the gaming industry, one thing remains the same—no fees, for anything. If you would like to be a member of Gamasutra.com, please join at the GDC show booth.

CMP Game MEDIA GROUP

continued from page 72
take ages before middleware is good enough and plentiful enough to satisfy innovation-hungry consumers. And since there isn't an abundance of middleware, most games using it are going to end up looking the same.

The Publishers' Point of View

Now let's look at the problems from the publishers' perspective, now that they can no longer afford the hit-and-miss approach of funding lots of projects with the hope that a quarter of them will result in high-quality, high-selling games.

First, they can fund games to the prototype stage, forcing the developers to prove themselves in game play and graphical terms in order to gain further cash. The problem is that it still costs around $200,000 to produce a prototype and it extends the development lifespan of any advancing project considerably.

Alternately, publishers can sign only talent with a proven track record. Such developers have been there and done it before, and are as safe a bet as you're going to get. But again, these people cost big money and you might get similar projects and stifle the innovation you crave.

Third, publishers can construct their own internal teams through creation and acquisition. This is a long-term answer with flexibility and growth potential. But will such a corporate culture lead to "safe" projects and a lack of creative hunger from these teams?

So How Can Developers Fund Games?

First, developers should fund their games not based on the music industry but rather the movie industry. Set up a company based on the game's intellectual property. Anyone can purchase equity in the whole package, including merchandise, film or TV rights, online properties, and other spin-offs. This extends also to marketing, distribution, and, well, everything to do with the concept.

Second, you can pool your talent to give smaller teams access to resources otherwise denied them. This isn't limited to technology, but can include everything from office resources to ideas. Lionhead's own satellite scheme works in this way, with the smaller groups benefiting from AI, 3D, and, just as importantly, testing and focus group access.

Another alternative is for software houses to get listed on the stock market. This is an extremely popular move at the moment, and can provide all the funds necessary for a team to take their game from conception to bagged and boxed without worry. You can get a higher royalty rate, too, by finding a publisher and distributor at the end of the project. You do of course need a track record, a stunning idea, and ultra-smart management, but if you can pull it off, you'll have the last laugh.

Finally, venture capitalism is also a viable alternative. Although you'll get locked in with the backers, you'll receive the money you need and you might not face the tough milestones a seasoned publisher would demand. The purse strings are held tightly but the men in suits want to see your game out there and selling, so they'll remain on your side if you can ultimately deliver the goods.

Our industry needs to face the fact that tomorrow's games *will* cost more. But games will still need the lifeblood of innovation, especially if we desire the holy grail of being truly mass-market. Both developers and publishers have to be innovative in another way, too. They must find alternative sources of investment and convince a great many people that their game is going to sit comfortably in the top five. For the games that ultimately achieve this kind of success, this is and will always be an incredibly profitable business.

AUTHOR'S BIO | *Peter Molyneux co-founded Bullfrog Productions in 1987 and created a new genre of computer games, the "god game," with the release of* POPULOUS. *Since then Peter has been responsible for a string of massive-selling games including* POWERMONGER, THEME PARK, MAGIC CARPET, *and* DUNGEON KEEPER. *He founded Lionhead Studios in 1997, whose first game* BLACK & WHITE *is due for release later this year.*

ADVERTISER INDEX

COMPANY NAME	PAGE
Adaboy	33
Alias|Wavefront	22
AICS	60
Apple Computer	C2–1
Autonomous Effects	69
Beatnik	C3
Capcom	65
Cibro Technologies	40
Cinram	69
Conitec	62
Criterion	5
Dice.com	66
Digimation	57
EA.com	61
Ensemble Studios	64
Game Refuge	66
Havok.com	2
Hewlett-Packard	17
IBooks.com	6
Intel Corp.	9
Interact Source	66
Licensemusic.com	55
LIPSinc	37
MathEngine	25
Motek	19
Multigen	10
Muse Corporation	66
Newtek	27
Numerical Design Ltd.	13
NxN Software AG	21
RAD Game Tools	C4
Rainbow Studios	68
Retro Studios	67
Skills Village	31
SN Systems	35
Unique Development	67
Vancouver Film School	68

Next-Generation Gaming

Raising the Stakes in the Publisher-Developer Relationship



The hype surrounding the next generation of games is at fever pitch. But with quality comes big expenses and long development times. Developers need money and publishers need products. Something's got to give.

Any number of seasoned game players knows about the next generation of hardware. They have a good idea of its speed, power, and versatility, and therefore not only expect but *demand* software to be worthy of it. The public wants ever more accessible, better-looking, more compulsive, more immersive, deeper games. And every development studio, big or small, is going to try to give them exactly what they want. Of course, this is exactly how it should be.

But there's a price to pay for these new, better-in-every-respect games. Unparalleled innovation costs dearly. It takes a lot longer to create software of this magnitude and it costs a lot more money. I have witnessed a great many game concepts recently and virtually all are attempting to implement brave, sometimes outstanding ideas. Some of these are original, some are born of licenses, but the one thing they have in common is that all are very expensive. And industry-wide, they aren't going to get any cheaper.

So how is funding currently arranged? A lot of existing publisher-developer contracts are based upon the music industry model. A publisher, perceiving greatness and predicting success in a group of talented individuals, hands over a wad of cash. The team then goes and creates a hit product. While this model has worked well (with a few exceptions in both the music and game industries), it's a risky prospect, and really only seen in cases where the game will be in development for less than two years and will cost less than $4 million to produce. The truth is, with the new games being conceived now, figures of three years and $10 million will become the norm.

The Problems Developers Face

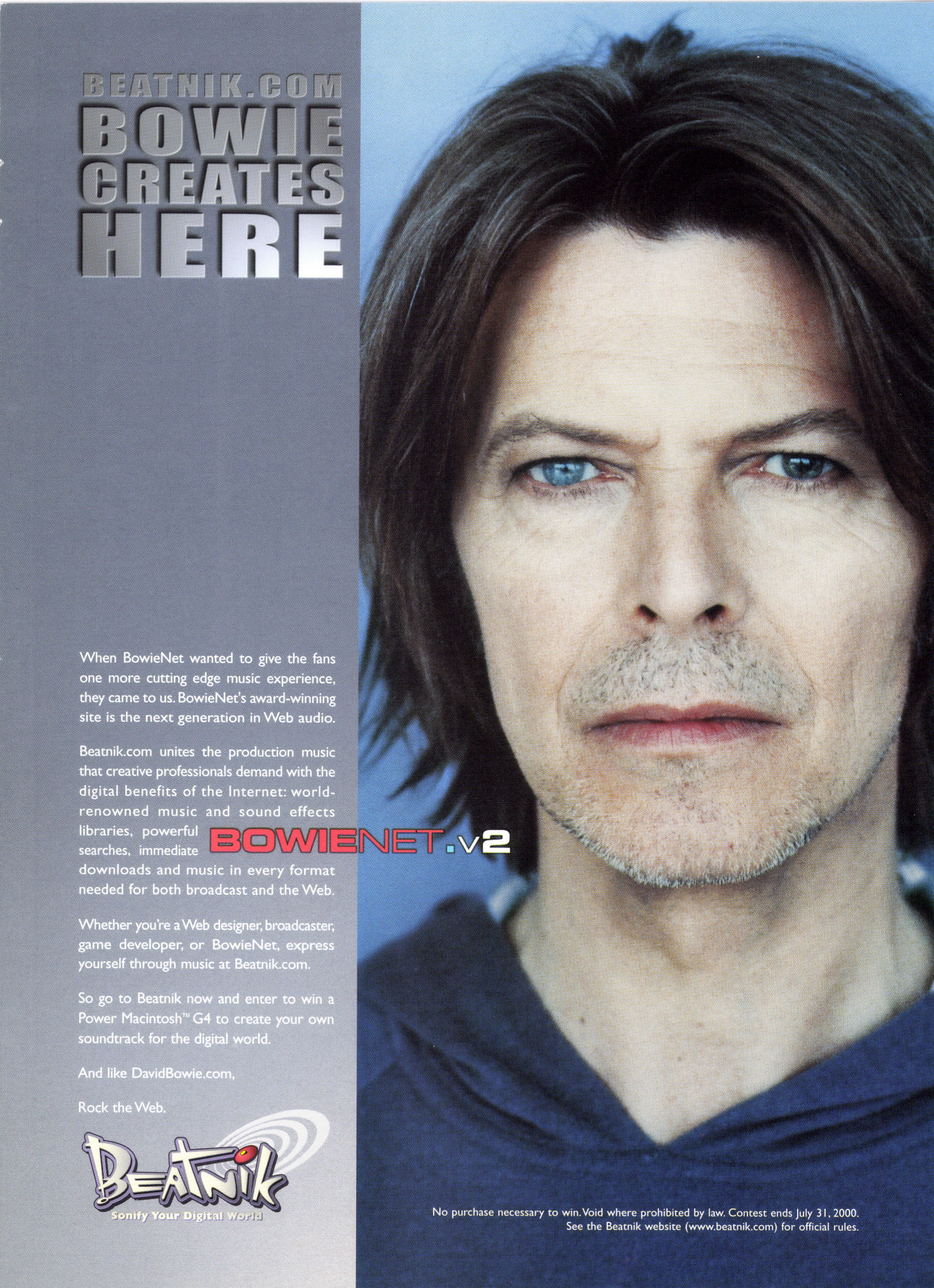
We're all starting to compete like crazy with each other in every aspect of game creation. This is good for the consumer, because it means we build games to a far higher specification than ever before. Admit it — in the U.S. and Europe, we view the latest screenshots from Japanese development houses with awe and mounting dread as we realize that the eyebrows on our main characters don't comprise individual hairs.

So competition is a good thing, even if it means we have to keep pushing the envelope until it rips. There are three recognized ways of keeping up the momentum. First, you can grow a team of geniuses. Talented people don't come cheap, and you'll be looking to lure the best from Disney for your animation and Pixar for your models. So we're not saving money here.

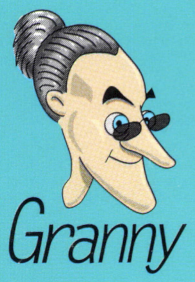
Second, you can form partnerships with out-of-house resources. Many studios are doing just this and relationships can thrive with animation houses or art or programming teams. Watch the spreadsheets, though. Budgets can easily creep upwards toward the gulp-worthy $10 million mark. In fact, there are some projects and game designs being talked about at the moment with figures of more than $20 million attached. If that kind of money makes you shiver, consider that as a rule, only the top five games at any one time make money.

Another option is to purchase existing game engines, animations, art, and other middleware development tools. This approach is the current flavor of the month: Why struggle to produce your own technology and artwork when you can dial out for it? O.K., but we're talking about the next generation of software here. It'll

continued on page 71


BEATNIK.COM
BOWIE CREATES HERE
When BowieNet wanted to give the fans one more cutting edge music experience, they came to us. BowieNet's award-winning site is the next generation in Web audio.
Beatnik.com unites the production music that creative professionals demand with the digital benefits of the Internet: world-renowned music and sound effects libraries, powerful searches, immediate downloads and music in every format needed for both broadcast and the Web.
BOWIENET.v2
Whether you're a Web designer, broadcaster, game developer, or BowieNet, express yourself through music at Beatnik.com.
So go to Beatnik now and enter to win a Power Macintosh™ G4 to create your own soundtrack for the digital world.
And like DavidBowie.com,
Rock the Web.
Beatnik
Sonify Your Digital World
No purchase necessary to win. Void where prohibited by law. Contest ends July 31, 2000.
See the Beatnik website (www.beatnik.com) for official rules.

The Best in Game Development Technology!



Granny Character Animation

Introducing Granny - the most sophisticated character animation system available. **Smoothly animate** your characters blending from a run to a walk to a sliding stop, all while using **real-time inverse kinematics** to point or look at other objects. **Deform clothing** to fit your different characters. Perfect exporting with built-in Bink **texture compression**! Much more!



Bink Video Technology

Bink - the only true-color codec good enough for games! Incredible quality (**better than DVD**/MPEG II). Up to **three times faster** than other true-color codecs. Perceptibly lossless, **8 to 1 audio compression**. Decompresses to DirectDraw, DIBSections, YUV overlays, or any other piece of memory. Available for Win32 (MacOS coming). Think Bink!



Smacker Video Technology

Smacker - the best 256-color codec on the planet! Smacker is the best codec for: **256-color games** (of course), games targeting Pentium 133s and below, games needing video sprites or video transparency (which is **much faster in 256 colors**), or games with very high-resolution videos (800x600 or 1024x768). Smacker is available on Win32, Win16, DOS, and MacOS.



Miles Sound System



Miles - the ultimate sound system! Miles supports 2D audio, 3D audio, MIDI with DLS, streaming, cd audio (red book), **DSP filtering**, **MP3 playback (including patent rights)**, **Internet voice chat** (2900, 2400, and 1200 bits per second codecs), Creative **EAX 1 and 2**, Aureal **A3D 1 and 2**, RSX 3D, DirectX 7, Dolby Surround, QSound, **software EAX emulation**, and more!

NO ROYALTIES WHATSOEVER! Over 3,000 games use our technologies!

Call today or visit our web site for more info:

425-893-4300 www.radgametools.com

